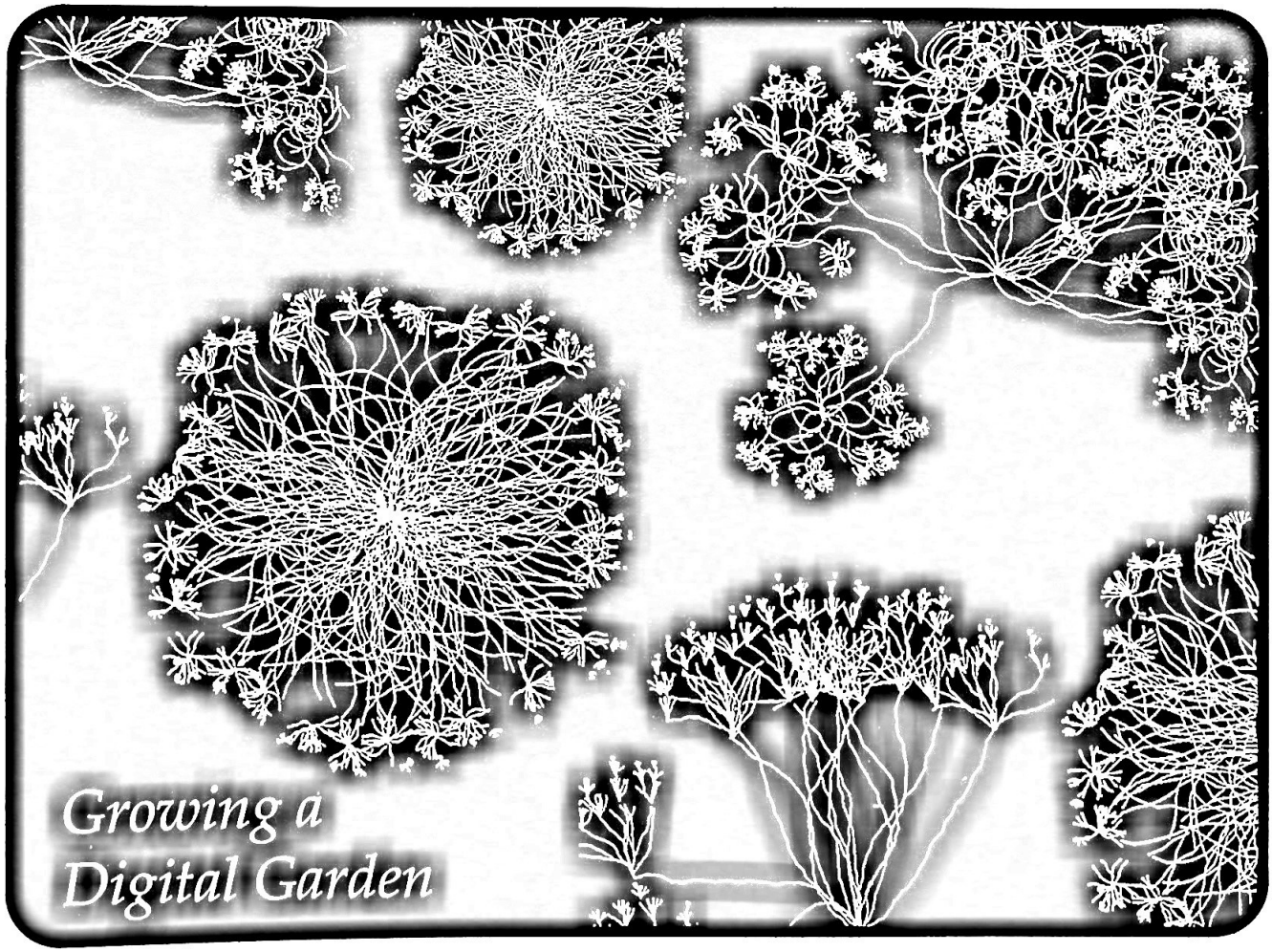
MAY/JUNE 1990

# Algorithm 1.4

THE PERSONAL PROGRAMMING NEWSLETTER




*Fox and Hare Dynamics*
*Fractal Clouds and Waves*
*Neural Nets that Learn*

# Cambridge University Press

## Numerical Recipes in Pascal
The Art of Scientific Computing
*Revised Edition*
***William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling***
The new Pascal version of this highly successful text and reference provides lucid, easy-to-read discussions of the most important practical numerical techniques for science and engineering. More than 200 actual, working Pascal procedures are ready to be included in the reader's own programs. Not simply translations from the FORTRAN and C procedures of previous versions in those languages, these 200 programs have been completely reworked in a clear, consistent Pascal style.

| | | | | |
|---|---|---|---|---|
| Pascal: | 1989 | 762 pp. | 37516-9 | Hardcover $47.50 |
| C: | 1988 | 768 pp. | 35465-X | Hardcover $47.50 |
| FORTRAN: | 1989 | 702 pp. | 38330-7 | Hardcover $47.50 |

## Introduction to Dynamical Systems
***D.K. Arrowsmith and C.H. Place***
This is an introduction to the mathematical structures underlying models of systems whose state changes with time, and which therefore may exhibit "chaotic behavior".
1990 350 pp. 30362-1 Hardcover about $79.50
31650-2 Paper about $25.00

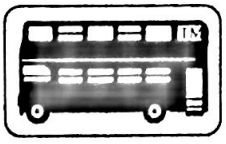
## Dynamical Systems and Fractals
Computer Graphics Experiments with Pascal
***K.H. Becker and M. Dörfler***
***Translated by I. Stewart***
This study of chaos, fractals and complex dynamics introduces the reader to an area of current scientific research that was scarcely possible until the availability of computers.
1989 350 pp. 36025-0 Hardcover $59.50
36910-X Paper $19.95

***Now in paperback...***

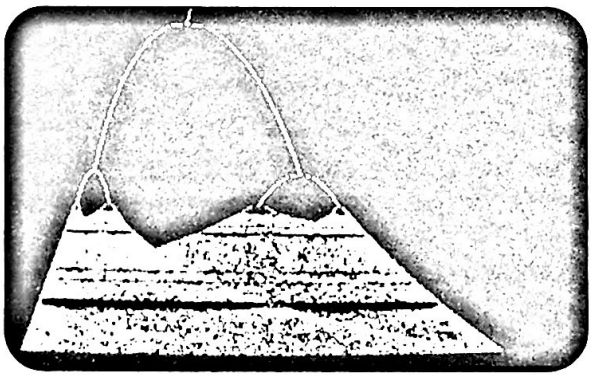
## Efficient Parallel Algorithms
***Alan Gibbons and Wojciech Rytter***
This largely self-contained text is an introduction to the field of efficient parallel algorithms and to the techniques for efficient parallelism.
1988 272 pp. 38841-4 Paper $24.95

At bookstores or order from
**Cambridge University Press**
40 W. 20th St., NY, NY 10011
Call: 800-872-7423, outside NY State. 800-227-0247, NY State only.
MasterCard/VISA accepted. Prices subject to change.

# EXPLOR CHAOS
with
# EXPLOR-I
from
## TURING OMNIBUS



P.O. Box 1456,
London, Ontario
CANADA N6A 5MZ



For PC and compatible computers
with CGA compatible graphics.

EXPLOR-I is a package of programs designed for amateur and professional explorers alike. Select up to eight programs for your own personal package: Julia and Mandelbrot sets, Wallpaper for the Mind, Exploring Chaos, The Invisible Professor, The Cyclic Space Cellular Automaton, Star Clusters Simulation, Computer Popcorn, Biomorphs. In addition, we now offer the Core War Coliseum, the official Core War programming and tournament package. Write for our free catalogue to-day.

# Algorithm

THE PERSONAL PROGRAMMING NEWSLETTER

Algostaff

Editor
A.K. Dewdney

Assistant Editors
Jennifer Connor
John O'Hagan

Managing Editor
Louis D. Magguilli

Layout & Design
Tammy Golinski

Production Services
Bogdan Graphics Inc.

ALGOLETTERS
*The Paideia School and Boids* ..... 2

EASY PIECES - Michael W. Ecker
*Foxes and rabbits that end up on a menu* ..... 4

COMPUTER ADVENTURES- A.K. Dewdney
*Four fractal elements: Earth, Air, Fire and Water* ..... 8

PERSONAL PROGRAMS - Clifford A. Pickover
*A random walk through a digital garden* ..... 12

ALGOPUZZLE - Dennis Shasha
*Dr. Ecco turns up some bad pennies* ..... 15

ALGOFACT - Edward A. Rietman and Robert C. Frye
*Neural networks that learn* ..... 17

SPECIAL REPORT - William R. Buckley
*Artificial Life in Santa Fe* ..... 22

ALGOFICTION - Tekind Weywadeen
*The Prisoner of Baghdad* ..... 24

BULLETIN BOARD
*Software, games and publications from individuals and companies* ..... 28

Address all editorial correspondence to A.K. Dewdney, 2 Oxford Street West, London, Ontario N6H 1P9, CANADA. Address advertising, business and all other correspondence to Louis D. Magguilli, Algorithm, P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Road S., London, Ontario N6K 1M6. Owned and published bimonthly by 854832 Ontario Inc., doing business as *Algorithm*. All rights reserved, including the right to accept or reject all editorial and advertising matter. Unsolicited materials must be accompanied by self-addressed return envelopes and postage or dollar equivalent. No part of this newsletter may be reproduced without the written permission of *Algorithm*. Copyright© 1990 by *Algorithm*. Mail subscriptions: $29.95 (U.S.) per year; single copies $5.00 (U.S.). Overseas add $4.00 (U.S.) for shipping. ISSN 0843-9753. Algophone (519) 432-8042. Mailed April 1990. Second Class Mail Permit Number 8561. Postage paid in London, Ontario.

# ALGOLETTERS

## *PLAYING (AND MISPLAYING) WITH ALGORITHMS*

Here at the Paideia School, a private school in Atlanta, Georgia, mathematical investigation goes on as a collaboration of students, teachers, and mathematically interested parents. Much of this occurs in the guise of playing with computers and algorithms. We have come across a number of interesting things, both intentionally and unintentionally, which I would like to share with the readers of *Algorithm*.

### *PLAYING WITH THE HENON MAP*

The Henon map was shown to us by Professors Fox and Harter at the Georgia Institute of Technology. The form they gave was

$$newx = y - x^2 \qquad (1)$$
$$newy = a - x$$

This has been written up in computer magazines; however, the form is usually given differently, using trigonometric functions. The game to play is to specify a starting $x$ and $y$ and a value for the parameter $a$. Iterate it, say, 300 times (see Algorithm 1), and the orbit of the initial point is produced (fig. 1). One then chooses more starting points, obtaining an image of the possible orbit behaviors under this mapping (fig. 2). The orbits vary from the simple to the chaotic, hence their attraction. Our goal has been to twist these equations in as many ways as possible.

First twist: change the $x^2$ to $x^3$ or $\sin x$ or any other function. There will be a range of a values where interesting patterns develop. Fig. 3 uses $\sin x$ in place of $x^2$.

Second twist: One of my students, Jonathan Manson (now at Harvard University), discovered this one. Instead of choosing a value for $a$, then varying the initial point to produce the picture, fix the initial point and vary $a$ over some interesting range of values. This gives a picture of the evolution of the orbits of one particular point as $a$ increases (see Algorithm 2). Fig. 4 shows the splitting of the orbits of the iterative equations $newx = y - \sin x^2$ and $newy = a - x$, where $a$ is varied from 0 to 4 and the starting point is (.176, .51). We call it "Jonathan's Tornado."

Third twist: We played with many variants of (1) by replacing the right side with anything we could think of. Most of the variations produced attractor-repellor situations, with the orbit spiraling towards a fixed point. The Henon-type maps were qualitatively different. We decided that this was because they were area-preserving. Mathematically, this means that the Jacobian equals 1. Jonathan and another of my students, Jack Bross (now at Berkeley), surprised us with some new area-preserving maps. One of these is

$$newx = f(y)$$
$$newy = a + x/f'(y),$$

where $f'(y)$ is the derivative.

These produced new and striking patterns, an example of which is Fig. 5 for which $f(y) = \cos y$, and $a = -1$.

### *GROWING FLOWERS, SOMETIMES INTENTIONALLY— SOMETIMES NOT*

Another well known iterative map is the Feigenbaum equation $newx = ax(1-x)$. This equation was my first contact with chaotic systems and I suspect it was for many others. I, too, has been described in computer recreation literature. The particular algorithm (Algorithm 3) I am interested in is the one that displays $x$, the fixed point, versus $a$. It is one that most dramatically demonstrates the bifurcation route to chaos. Fig. 6 shows this graph and its important features. We gave a name to these pictures. We called them "flowers."

One day, the aforementioned Jack Bross came up with an unusual picture with two "flowers" side by side. Here is what he did. He replaced $r$ with a function that had the same range of values (0-4 in this case) but oscillated back and forth through this range. He replaced $a$ with $4 \sin a$. His iteration was $newx = 4 \sin (r)\ x(1-x)$. The result is in Fig. 7 (it is periodic in the horizontal direction). Other interesting pictures can be generated creating a function of $a$ that "tickles" interesting values of $a$.

Simultaneously, Bo Garrett, now at Duke University, came up with a similar multi-flowered graph (Fig. 8). His equation was

$$newx = x^2 - a^2x - ax + a.$$

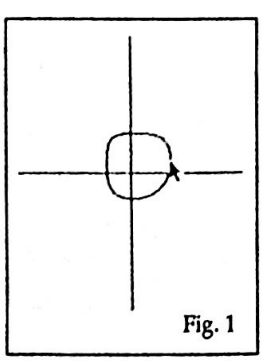


Fig. 1

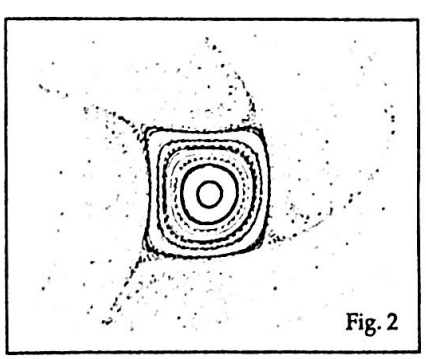


Fig. 2

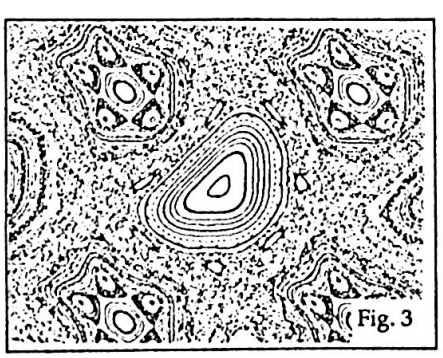


Fig. 3

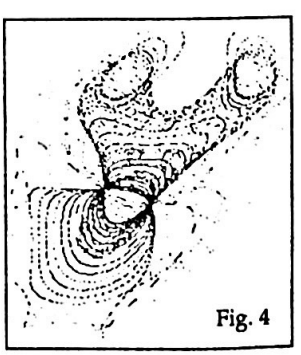


Fig. 4

A mistaken algorithm: One of my Algebra II classes was graphing equations on the computer. The graphing algorithm was a normal one: choose *x*, find *y*, plot point, repeat until sated. We were graphing ordinary stuff— parabolas, cosines and such. Ted Thompson called me over. Fig 9 was on the screen. I became really excited. Two "flowers" were on the screen. The single graph turning to two graphs, then to 4, to 8, and to chaos was there, but this time it was present in two mirror reversed patterns. Here is what Ted had done. He had graphed the "equation" $y = x*cos(y)$. In BASIC the "=" means "replace" so that the y values are not the same on both sides of the equation. On the left it is the new *y* and on the right it is the old *y*; hence this mistaken algorithm becomes an iterative equation.

Steve Sigur
The Paideia School
1509 Ponce de Leon Ave.
Atlanta, GA 30307

**Algorithm 1**

```
Input a
Input x, y
repeat 300 times
    newx <- y - x^2
    newy <- a - x
    x <- newx
    y <- newy
    plot (x, y)
```

**Algorithm 2**

```
Input x, y
for a <- 0 to 4 step .1
    repeat 300 times
        newx = y - y^2
        newy = a - x
        x  = newx
        y = newy
        plot (x, y)
```

**Algorithm 3**

```
x = .1
for a <- 0 to 4 step .01
    repeat 20 times
        x = ax (1 - x)
    repeat 100 times
    x = ax (1 - x)
    plot (a, x)
```

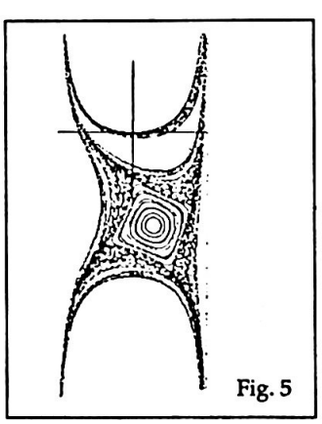


Fig. 5

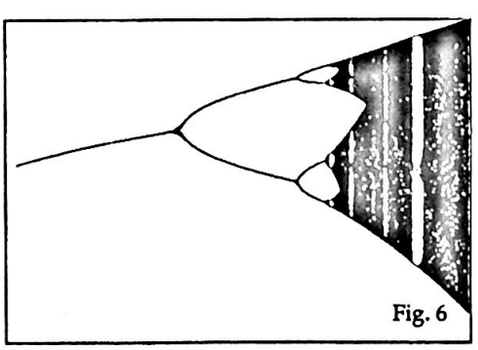


Fig. 6

## *BOIDS OF A FEATHER*

*Algorithm* 1.1 described a simulation of flocking birds ("boids"). I decided that it was possible to develop a simplified version of the program.

I started with the CLUSTER program from Computer Recreations, *Scientific American*, 1/86. The two programs use the same loop structure to calculate the accelerations, velocities and positions of the stars or boids (I constrained the boids to fly only in the XY plane). The major difference is the nature of the forces acting on the boids.

The essential feature of those forces is attraction at large distances and repulsion at small. This causes the boids to "flock" together. I used an inverse-square-law attraction and an inverse-cube-law repulsion. The interaction of boids and objects in the environment is similar, except that there is no attraction. For consistency, I used an inverse-cube-law repulsion here also.

I also added a constant force to each boid in order to entice the flock to fly across the screen. In addition, I added friction in order to prevent oscillations (obviously unrealistic behavior for real birds!).

The resulting program produces boid behavior very similar to that described in *Algorithm*.

Peter Winkelstein
392 Capen Blvd.
Amherst, NY 14226

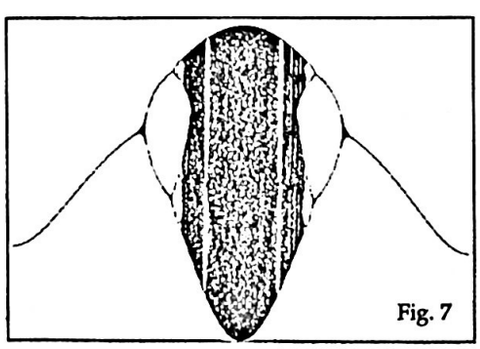


Fig. 7

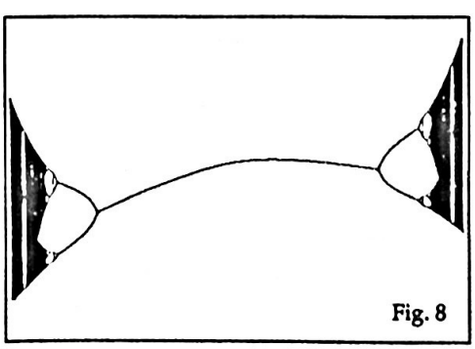


Fig. 8

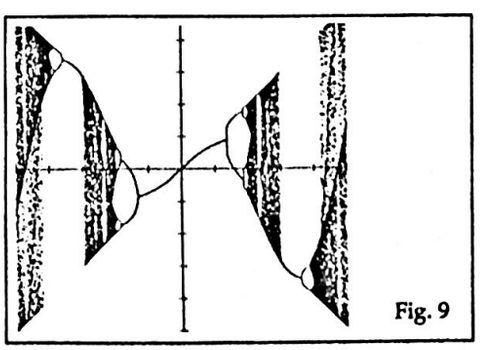


Fig. 9

# EASY PIECES

by Michael W. Ecker

## *Foxes and rabbits that end up on a menu*

In our ongoing pursuit of elementary graphic commands, it seems reasonable to graph the results of foxes chasing rabbits. Later, everything we've done can be put on a menu that you can create for yourselves.

In the previous Easy Pieces, I introduced commands that produce lines. It is time now to put this and other graphic commands to good use. We'll illustrate the old cliche about a picture being worth a thousand words by tracing visually the results of a struggle for existence between a population of foxes and another population of rabbits. I am indebted to Dolores Garcia Garcia of Spain who sent me a version of the program I call PREDATOR. Besides being an amusing exercise in programming, PREDATOR creates behavior that has a firm foundation in the theory of predator-prey relations. In fact, PREDATOR will graph the number of rabbits and foxes present at each cycle of computation.

First, we need to extend our range of graphic powers by adding a new command: PSET. This command, which uses the syntax PSET (X,Y), illuminates the pixel at position (X,Y) on the screen, measured from the upper left. Note the difference from the position we would normally assume for the origin, at the lower left or, for that matter, in the middle. One consequence of the origin being in the upper left of the screen is that the y-axis points down instead of up. If for some reason you find this too confusing, you can effectively change the origin to its more "normal" position by using an old trick: Suppose your screen is 640 pixels across and 200 pixels high. Instead of plotting the point (X,Y), you can plot (X,199-Y).

As with the LINE command, PSET is available on PCs that have color screens. What about monochrome? I know of a company that distributed a reliable color-simulator for monochrome screens, but the company seems to have gone out of business. I would be grateful if any readers can steer us to similar software that is still available.

We can use the PSET command to draw a line, as well as to draw the curves that result from PREDATOR. Recall that a non-vertical line with slope m and passing through the point (0,B) has the equation

$y=mx+b$

Can you write a simple program that will draw $x$ and $y$ axes, allow input of $m$ and $b$ and then draw the line? What restrictions will there be on $m$ and $b$? How much of the line should your program draw?

Another way to draw a line is to use the two-point form. If the user inputs two points, $(x_1, y_1)$ and $(x_2, y_2)$, your program could draw a straight line from one point to the other by stepping $x$ from $x_1$ to $x_2$ and, for each value of $x$ in between, plot

$y = m(x-x_1) + y_1$,

where $m$ is the slope of the line, namely, $(y_2-y_1)/(x_2-x_1)$.

Whichever method you use, you will see a big difference between the speed of either program and the LINE command.

### *FOXES AND RABBITS*

Imagine that we are looking at a small patch of environment that is home to both foxes and rabbits. Other species may live in this environment, but we will ignore them. We will also assume that the actual numbers of either animal present in this pretend environment at any one time are within range of the screen coordinates. We are reasonably safe, in other words, if neither population goes above 200 in the case of a 200-by-640 screen. Do not be upset if the program PREDATOR produces fractional individuals of either species. It is an approximate program, like many mathematical models.

The actual numbers of foxes and rabbits depends on the initial numbers of each, at least in the short run. In the long run, the numbers depend on the birth and death rates that one assigns to each species. The numbers can oscillate or settle down to a steady state, if one exists. The whole story is told by the steadily unfolding screen graphic of PREDATOR.

The box below shows a simple algorithm on which the PREDATOR program is based. First, the program sets up the screen and prints an announcement of its presence. Next, it asks the user to input values for seven variables to be used by formulas inside the program. These variables include the birth and death rate of the rabbits, the birth and death rate of the foxes, as well as the rate at which foxes eat rabbits and the number of each species that is initially present. In the next step, PREDATOR draws a graphic box that will enable a user to make sense of the data that the program will subsequently plot.

In the final step, the program

**Algorithm for PREDATOR**

1. initialize the screen
2. prompt user for initial data
3. draw box for population graphics
4. calculate next populations
5. plot population point
6. go to step 4.

plots population points. Here the number of foxes and the number of rabbits together yield the coordinates of a point that summarizes at a glance, so to speak, what is going on in our miniature ecosystem. As the two populations change, the point wanders around the screen. Sooner or later, a pattern shows up. The loop that I have used is the simplest possible kind. The program continually goes back to step 4 to plot the next population values.

The program I am about to present is similar to the one Ms. Garcia Garcia has sent. I've made a few changes to simplify coding. The box below shows the program I call PREDATOR. Lines 10, 20 and 30 correspond to step 1 of the algorithm. Lines 40 to 100 comprise step 2. Lines 110 to 130 amount to the drawing and labelling of an axis-box, and the remainder of the program corresponds to the last two steps of the algorithm.

In more detail, readers may recall my remarks in the previous Easy Pieces column about the various screens. Here, we put screen 2, the intermediate resolution screen, into play. The program then clears the screen and prints its title. Lines 40 to 100 are all INPUT statements. The user is informed, as each line is executed, what variable is to be inputted.

Line 110 clears the screen to wipe away all the prompts and replies previously printed. This clears the decks for action. The remainder of this line is a command that prints the word "foxes" at the location (3,5), near the top, left-hand side of the screen. The next line, 120, prints the word "rabbits" to label the axis and the next line, 130, draws a box around the screen.

The next two lines, 140 and 150, contain two formulas, one for the number R of rabbits and the other for the number F of foxes. In this simulation, we don't have graphic foxes eating graphic rabbits. Rather, a tendency of foxes to eat rabbits is expressed by a formula discovered in the 19th century by the Italian mathematician, Volterra.

Line 160 tests whether the current values of F and R are within bounds for the screen. If they aren't, execution simply jumps back to line 140. Line 180 draws the point that represents the current state of the two populations. The next three lines write out the current numbers (for anyone quick enough to actually read them) and the last line, 220, directs execution back up to line 140.

There is no way out of the loop thus constructed except to type control-c or control-break and restart the program. There are more graceful loops, of course. I have explained how to do this in Easy Pieces 1.1 and 1.2 but, just as an exercise, I will add some hints on how to do this in the present case. Use a variable called COUNT that is initially set to 0. Every time a new point is plotted by PREDATOR, add 1 to COUNT and test whether it has reached, say, 500. If it has, transfer execution around what is currently the last statement to a new last statement that simply says END.

With so much given, what reader can resist the temptation of typing in PREDATOR and giving it a whirl? To see typical behavior, try the following values: A=.04, B=.00005, C=.002, D=.03, E=.0002. For initial numbers of foxes and rabbits, use R=300 and F=30. In fact, numbers in these neighborhoods make useful starting values for many of the simulations you may want to try. Those who want to see some strange behavior in the two populations may

**The program PREDATOR.BAS**

```
10  KEY OFF: SCREEN 2
20  CLS: PRINT "SIMULATION OF FOXES AND RABBITS"
30  PRINT
40  INPUT   "BIRTH RATE OF RABBITS"; A
50  INPUT   "DEATH RATE OF RABBITS"; B
60  INPUT   "EATING RATE OF FOXES"; C
70  INPUT   "DEATH RATE OF FOXES"; D
80  INPUT   "BIRTH RATE OF FOXES"; E
90  INPUT   "INITIAL NUMBER OF RABBITS"; R
100 INPUT   "INITIAL NUMBER OF FOXES"; F
110 CLS: LOCATE 3,5: PRINT "FOXES"
120 LOCATE 24, 70: PRINT "RABBITS"
130 LINE (2,0)-(639,199),,B: LOCATE 1, 60
140 R=(1+A-B*R)*R-C*P
150 F=(1-D)*F+E*P
160 P=F*R
170 IF NOT ( F>0 AND R>0 AND F<200 AND R<640 THEN 140
180 PSET (R,199-F)
190 LOCATE 1,20
200 PRINT   "FOXES = ";: PRINT USING "###.##";F;
210 PRINT   "RABBITS = ";: PRINT USING "###.##";R
220 GOTO 140
```

try the following values: A=3.1, B=.01, C=.03, D=.5, E=.0058. I wonder whether anyone will be able to discover what combinations of parameter values lead to stable (although oscillating) behavior, and which values lead to a spiral of extinction. Readers of *REC* (see the Bulletin Board ad on page 28) will have already seen a somewhat fancier version of PREDATOR.

### *WHAT'S ON YOUR MENU?*

Now that we're starting to accumulate some programs, here's a nifty method for creating your own menu system. You may use it to keep frequently used programs ready to hand. There are two ingredients to this automation process:

1) creating the menu program,
2) having the program always called every time you enter BASIC or whenever you quit one of the programs already on the menu.

Let's say the programs are HAILSTNE.BAS from Easy Pieces 1.1, ORACLE.BAS from Easy Pieces 1.2, and LINES.BAS, LITNLINE.-BAS, both from Easy Pieces 1.3. Now, of course, you can add PREDATOR. BAS from the present column. I hope you've been saving all your programs!

The menu program, called MENU.BAS, is shown in the box on the right. In principle, the program is simple enough. It uses a string array called PROGRAM$ that, in line 20, is dimensioned to LAST. This means that the current value of the variable LAST, 5 at present, will be used to tell your BASIC system how many entries the array will have. If and when you add other programs to your menu, always change the value stored in line 20 to reflect the new total.

At the screen location (3, 30), the menu program text types its heading, then in line 50, leaves two blank lines (by printing nothing) for what is to follow. A loop that starts on line 60 and ends on line 90 systematically reads each of the programs currently stored in the menu array PROGRAM$ onto the screen where it prints the title and number of each program.

In line 110, you are prompted to type the number of the program that you want to run. The number is stored in the variable X. Line 120 tests to see whether the number requested is reasonable or not and line 130 actually causes the program with that number to be run. Everything from this point on will proceed just as though you had typed all the usual commands to get that particular program up and running. But notice how much easier this is! The last line of the program is a data statement that holds the names of all the programs you currently wish to save. This line must be changed, of course, every time you add a new program to your menu.

Save the program as MENU. BAS in the usual fashion. By now, you should have added what might be called a "graceful ending" to each of the programs you have saved. If they are endless loops (like PREDATOR, above) then you will be unable to get the menu program back automatically when you exit from BASIC.

In my next column I will begin to respond to some of the reader mail that has been piling up. I will be happy, as usual, to look at special responses to this and past columns as well to gather suggestions for future topics. Programs on paper or disk (stored as an ASCII file, please) are both welcome. Write to me at the following address. If you would like a reply, please enclose a large self-addressed envelope accompanied (for US residents) by a stamp.

*Dr. Michael W. Ecker*
*"Easy Pieces" Columnist,*
*ALGORITHM*
*129 Carol Drive*
*Clarks Summit, PA 18411 USA*

Until next time, may your programming pieces be easy!

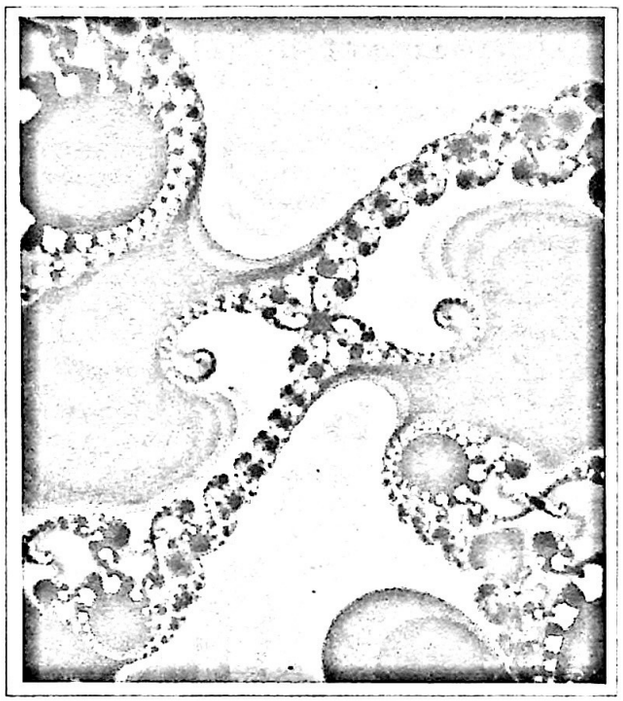
**The program MENU.BAS**

```
10  CLS
20  LAST= 5:  DIM PROGRAM$(LAST)
30  LOCATE 3,30
40  PRINT "MENU OF PROGRAMS"
50  PRINT:  PRINT
60  FOR NUMBER = 1 TO LAST
70  READ PROGRAM$ (NUMBER)
80  PRINT NUMBER; PROGRAM$(NUMBER)
90  NEXT NUMBER
100 PRINT:  PRINT
110 INPUT "WHICH PROGRAM (NUMBER) DO YOU WANT?"; X
120 IF X<1 OR X>LAST THEN 110
130 RUN PROGRAM$(X)
140 DATA "HAILSTNE", "ORACLE", "LINES", "LITNLINE", "PREDATOR"
```

Andromeda Research, Inc.

Presents the Ultimate Fractal Software
For the IBM® PC and Compatibles

# Fractal FX



(Formerly titled Fractal Explorer)

Only one fractal program offers:
- Almost unlimited zooming capability ($10^{72}$)
- Full screen and quadrant displays
- High-speed image generation—up to 100 times faster than the competition
- Sophisticated edge detection algorithm
- Professionally designed user interface
- Support for many high-res VGA cards (up to 1024 x 768) in both 16 and 256 color modes

Minimum Requirements:
- 8088 Microprocessor
- EGA or VGA Graphics
- Color Monitor
- 640K RAM
- Hard Disk
- MS DOS® 3.0 or later

Send $29.95 US (check or money order) to:
Andromeda Research, Inc.
6441 Enterprise Lane
Madison, WI 53719
or call 608-833-3022
VISA or Mastercard accepted.

# HOW TO READ ALGORITHMS

An algorithm is a generic description of a specific computation. The description does not follow the syntax of any particular programming language but it follows the overall form of a program; it consists of a sequence of informal statements. Even in our algorithms, there is some syntax, however. For example, assignment statements such as *newy* <– *a* - *x* use the left-arrow as a dramatic indication of how assignment statements actually work: the current values of variables that appear on the right of the arrow are combined according to the expression there (in this case, *a* - *x*) and the resulting number is placed in the variable (in this case *newy*). In most computer languages the left-arrow would be replaced by an equal sign (=) or colon-equals (:=).

Another feature of our algorithmic style is the use of indentation to set off blocks of instructions. Such blocks appear inside loops or conditional statements. The following algorithm contains four indented blocks:

```
input corna, cornb
input side
for i <– 1 to 100
    for  j <– 1 to 100
         x <– corna + j x side/100
         y <– cornb + j x side/100
         c <– int (x² + y²)
         if c even
              then plot (i, j)
```

The first block consists of the two input statements and the for-loop. Inside the for-loop is a second for-loop comprising the second block. Inside the second for-loop is a block of four instructions that are only carried out while the inner for-loop is executing. In some programming languages, such blocks are marked off by "begin" and "end" statements. In BASIC, a for-loop is demarked by a "for" at the beginning of the loop and a "next" at the end. The fourth block in this example consists of the plot instruction. This is indented because it is only executed when the condition (in this case, "c even") happens to be true.

In the informal algorithmic style, special commands can be invented and used, provided that they correspond to something a computer can do. For example, "int ($x^2 + y^2$)" means, "take the integer part of the sum of $x^2$ and $y^2$." sometimes the shorthand is extreme: a single, informal line actually corresponds to a whole block of lines. (Readers who want to see something mildly astonishing might enjoy converting the algorithm herein to an actural program).

# COMPUTER ADVENTURES

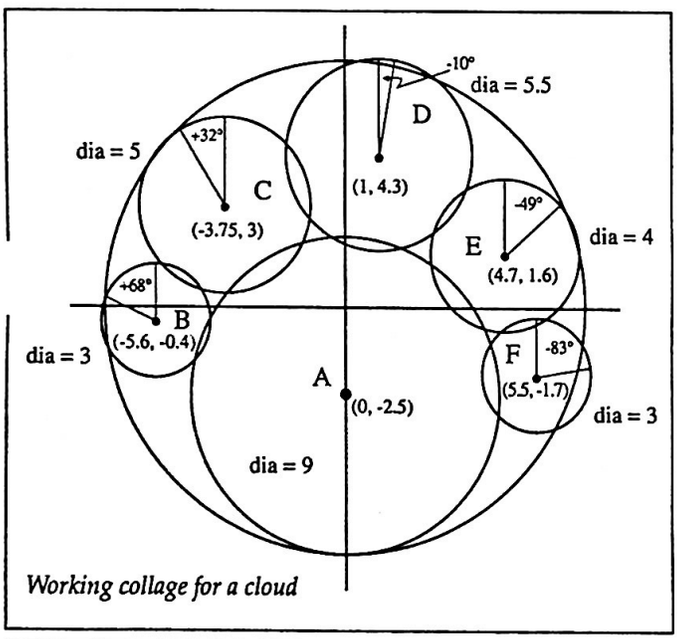
by A.K. Dewdney

## *Four fractal elements: Earth, Air, Fire and Water*

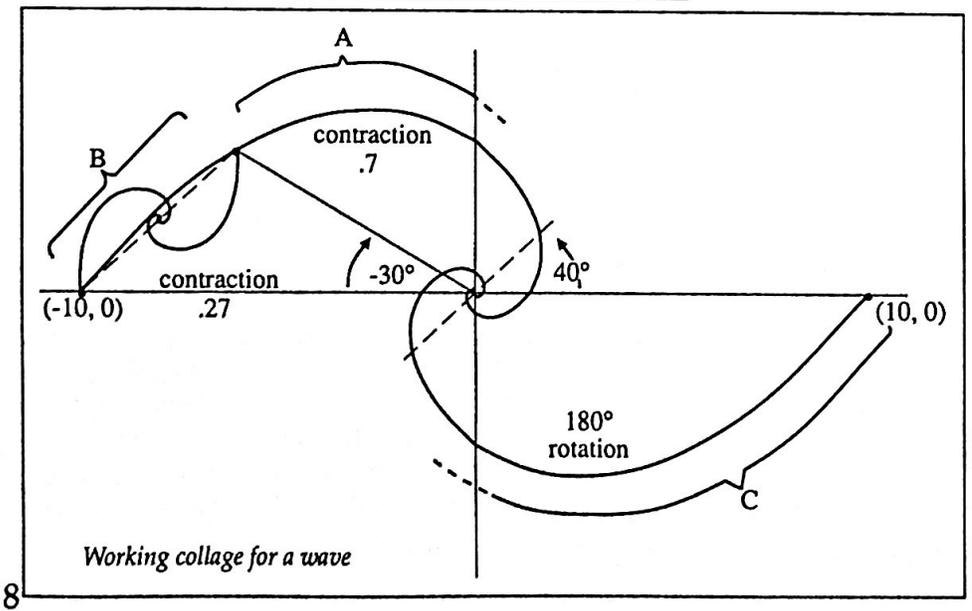
Since Benoit B. Mandelbrot first coined the term, we have learned to look for fractals everywhere, especially in nature. Although nature contains no true fractals, in which every part contains an infinite regress of similar parts, it abounds with structures that produce the illusion. One may find self-similar structure in certain leaves and plants, in shorelines, perhaps even in the structure of galaxies and galactic clusters. But a column on the elements of fractal generation may as well practice on the traditional elements of nature: earth, air, fire and water.

Earth may be symbolized by a jagged mountain, air by a puffy fractal cloud, fire by a fractal flame and water by an infinitely curly wave. Air and water are shown in the illustrations on subsequent pages, but readers can supply earth and fire for themselves, based on what they glean from this article.

In 1986 a technique for generating arbitrary fractals was discovered by Michael F. Barnsley, a mathematician at the Georgia Institute of Technology. The technique involves what Barnsley and his co-worker, Alan D. Sloan, call iterated function systems, or IFS for short. A set of simple equations and an iteration rule that uses them sends a point hopping madly about the plane. If the point is plotted in each of its successive positions, thousands of them in fact, a fractal image slowly fills in. Readers may have seen the results of iterated function systems in an article by Barnsley in *Algorithm* 1.2. More recently, my Mathematical Recreations column in *Scientific American* magazine carried a detailed description of how iterated function systems to generate fractals. In the present article, readers will find an algorithmic description that is complete enough to begin programming: you may float away on a fractal cloud or ride a fractal wave. Such is the power of computers!




*Working collage for a cloud*

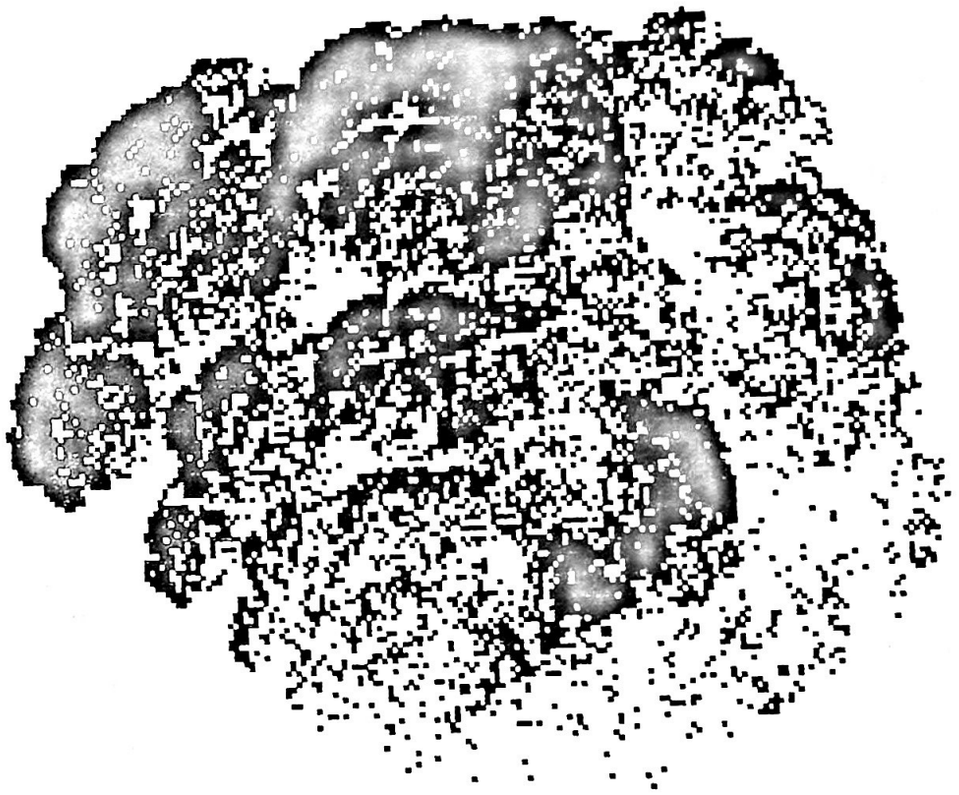



*Working collage for a wave*

### *A FRACTAL CLOUD*

Suppose you have a description of a fractal object (whether real or fanciful) in the form of a map. The object will normally be composed of parts, each similar to the whole. The first step in setting up an iterated function system, or IFS, is to prepare by hand what Barnsley calls a "collage." This is essentially a map of the object as a region which is divided into sub-regions that are similar to the region as a whole. Consider, for example, the upper collage shown on this page. Here, the region that represents the circular outline of a cloud as a whole has been subdivided into six sub-cloud regions.

The next step in the preparation of the program is to set up what mathematicians call affine transformations, one for each part of the whole. In the case of the cloud and its six sub-clouds, six affine transformations are called for. Each transformation has three parts: A shrinkage of the original cloud down to the size of the sub-cloud, a rotation of the sub-cloud, then a translation (shift) of the sub-cloud into its target position. The effect of such a transformation is to move any point in the big cloud into the corresponding point of the sub-cloud. Thus a point near the top of the big cloud will end up close to



the top of the sub-cloud.

To set up an affine transformation, you must first determine a reasonable coordinate system, one that makes the transformations simple. By placing the origin at the center of the big cloud, the transformations are easy to obtain, especially if the exercise is carried out on squared paper. At the end of this article, I will explain exactly how I obtained them. The general form of an affine transformation is:

$x' <-- ax + by + e$ and

$y' <-- cx + dy + f$

The values of $a$, $b$, $c$, $d$, $e$, and $f$ are best presented in a table. For the cloud the table has six rows, one for each sub-cloud in the collage. The sub-clouds are labelled with capital letters that have nothing to do with lower case letters just mentioned. For each sub-cloud, naturally, there is a corresponding transformation that maps the whole cloud into that particular sub-cloud:

| | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| A | .643 | 0 | 0 | .643 | 0 | -2.5 |
| B | .08 | -.199 | .199 | .08 | -5.6 | -0.4 |
| C | .303 | -.189 | .189 | .303 | -3.7 | 3.0 |
| D | .386 | -.068 | .068 | .386 | 1 | 4.3 |
| E | .187 | .754 | -.754 | .187 | 4.7 | 1.6 |
| F | .026 | .213 | -.213 | .026 | 3.5 | -1.7 |

It is a simple matter to produce two equations from each line of the table. Simply substitute the values of $a$, $b$ and e into the first equation and the values of $c$, $d$ and $f$ into the second one. I have used the algorithmic replacement symbol (<--) in place of an equals because that is how a fractal-generating program must ultimately use these numbers. Incidentally, I have placed the symbols $x'$ and $y'$ on the left-hand sides of these equations instead of $x$ and $y$. The reason lies in the preservation of information during a computation. If I used $x$ and $y$ instead of $x'$ and $y'$, the program would destroy the old value of $x$ before the second equation got to use it. Once both equations have been computed, the program may replace old values by new values by using the following simple assignments:

$x <-- x'$

$y <-- y'$

In any event, the coordinates $(x, y)$ represent a point before it is moved and the coordinates $(x', y')$ represent the point after a particular transformation has moved it.

Once we have set up the transformations for an IFS, we may set a point in motion by an iterative process; wherever the point happens currently to be, the process will plot it, then move it to a new location within the cloud as a whole. In the collage that we will use, the sub-clouds are represented by circles of different sizes. In most IF systems, the iterative process sends a point in the object as a whole into a sub-object with a probability that corresponds to the sub-object's area. In the case of the cloud, it turned out (by calculating the area of each circle, then reducing all the numbers proportionally until they added to 1) that the resulting probabilities must be:

A: .47, B: .06, C: .15,

D: .18, E: .09, F: .05

Unfortunately, the use of these probabilities results in a cloud that looks too bottom-heavy. To compensate, I greatly reduced the probability of a point landing in the large sub-cloud at the bottom of the collage. Giving cloud A a probability of .21 instead of .47, the result was definitely more cloud-like. Readers should feel equally free to tinker with the probabilities to obtain different effects.

At the toss of an abstract coin, the program selects one of the sub-clouds and maps the current point $(x, y)$ into it by means of the appropriate transformation. Each of the transformations shrinks the large cloud down to the size of the sub-cloud, rotates it slightly one way or the other, then translates it into the right position.

The basic IFS algorithm is simple and short enough to state right here and now:

1. $x <-- 0, y <-- 0$
2. for $n <-- 1$ to 20,000
   Z <-- random (A,B,C,D,E,F)
   $(x,y) <--$ Z$(x,y)$
   if n > 10 then plot (x,y)

On this basis, readers may construct the program I call CLOUD. In the first step, the variables x and y are set to 0. The second step involves a loop that repeats the basic iteration process 20,000 times. Readers are free to change this number if the images that result look either too dense or too light.

Within the loop, I have condensed

what will ultimately be many program instructions into single lines. For example, the "instruction" Z <-- random (A,B,C,D,E,F) really means, "Choose one of the six transformations A, B, C, D, E, or F based on their relative probabilities of .21, .09, .22 and .27, .13 or .08." How is this done? Imagine a one-unit line segment that is subdivided into consecutive lengths equal to these probabilities. The first division point would lie at .21, the second at .30 and the third at .52 and so on. In each case, merely add the current probability to the ongoing sum.

If a random number between 0 and 1 is chosen, it must lie somewhere on the unit line. The segment that it happens to fall into will be the basis on which the program CLOUD chooses which transformation to apply next.

A simple algorithm for this process would involve six separate tests. I have displayed the first three below so that readers will get the general idea:

1. *r* <-- random
2. if *r* < .21 then transform A
3. if *r* > = .21 and *r* < .30
   then transform B
4. if r > = .30 and *r* < .52
   then transform C

etc.

Even this algorithm is not quite complete. What, after all, do the commands "transform A," and so on, mean? The structure that I have chosen for CLOUD must expand each of these commands into little algorithms of their own, algorithms that correspond, in fact, to the second line of the loop.

To actually carry out transformation B, for example, requires two steps:

1. *x'* <-- .08*x* - .199*y* -5.6
2. y' <-- .199*x* + .08y -0.4

These instructions will look a little friendlier to readers who glance at the second line of the IFS tableon page 9: The numbers .08, -.199, .199, .08, -5.6 and -0.4 appear in the second row, the one for transformation B.

Each of the four algorithmic modules "transform A," "transform B," and so on, must be translated into similar two-line algorithms, each followed by the assignments of *x'* and *y'* back into *x* and *y*.

In the final stage of the loop, the point (*x*, *y*) must be plotted. This is not quite as simple as saying, "plot (*x*, *y*)." Indeed, the programmer of CLOUD must use an appropriate screen coordinate system as pointed out by Michael Ecker in Easy Pieces 1.3, for example. But suppose that your display screen happens to be 200 pixels wide and 100 pixels high. The coordinate system used in the four transformations described above places the origin in the middle of the cloud. But in most computer screens, the origin lies on the left side of the screen, either at the top or the bottom. How do you "plot" the point (*x*, *y*)?

The answer lies in using two systems of coordinates, one for computation of the cloud, the other for display purposes. The first system of coordinates is used for a virtual cloud, so to speak, that we never see. The second system of coordinates is used strictly for display purposes. Thus, every time a new point (*x*,*y*) in the virtual cloud is computed by the program, it must convert these coordinates into a new point (*w*,*z*) in the display cloud.

The virtual cloud happens to be 14 units (of squared paper) in diameter. To create a display cloud that is, say 112 pixels in diameter, the virtual coordinates must each be multiplied by 8. But that is not enough. They must also be translated from the left edge of the screen over to the middle. The following statements will handle the case of a 20- by-100 screen that has its origin in the upper left corner. The variables *w* and *z* represent the display coordinates.

1. *w* <— 8x + 100
2. *z* <— 8y - 50

This completes the description of CLOUD. I will not claim for a moment that I have selected the best possible collage for my cloud. Others may find a better arrangement of sub-clouds that fill the overall shape in a more natural or aesthetically exciting fashion. In general, the more open space within the collage that is not covered by sub-figures, the more open and diffuse the resulting graphic image will be.

## *A FRACTAL WAVE*

Our second IFS fractal represents the element water. This time I will use not a two-dimensional collage but a one-dimensional one. The second figure on page 8 shows a pair of spirals joined together at their apexes to produce a shape that, overall, looks somewhat like a wave. The shape may be regarded as a one-dimensional "region" that may be sub-divided into spheres of influence for three separate transformations, A, B, and C.

The first transformation, A, maps the entire double-spiral wave into a smaller version of itself still centered on the apex. This transformation not only shrinks, but it rotates the figure so that it still fits the smaller portion of itself. The second transformation maps the entire wave into the small wavelet labelled B in the figure. Readers will see at once that this is simply a smaller version of the entire double spiral placed at the trailing edge of the large wave.

But what about the other half of the wave, the trough into which it threatens to crash with a fractal roar? The third transformation, C, simply rotates the entire figure by 180 degrees so that whatever gets generated on one side of the image has a chance of being mapped over to the other side

in order to complete the picture. In fact, because the right half of the wave has half the area, the third transformation gets half the action. The probability with which the program I call WAVE uses transformation C is exactly .5. The table below shows the IF system I

| | a | b | c | d | e | f | p |
|---|---|---|---|---|---|---|---|
| A | .606 | .35 | -.35 | .606 | 0 | 0 | .35 |
| B | .207 | -.174 | .174 | .207 | -7.9 | 1.6 | .15 |
| C | -1 | 0 | 0 | -1 | 0 | 0 | .50 |

developed to produce the wave image. Readers are free to write their own version of WAVE, using these equations or others they might wish to derive for themselves.

## *A FFINE METHOD*

This raises the question, of course, "Just how do you find those equations?" Insofar as they involve only contractions, rotations and translations, they can be calculated on the basis of measurements taken directly from the figure you have drawn on squared paper.

Consider, for example, the sub-cloud B in the cloud collage on page 8 Its diameter, as I measured it on squared paper, was 3 units while the cloud as a whole had a diameter of 14 units. This meant that the program would have to shrink the large cloud by a factor of 3/14 or .214. Next, I decided that the small cloud should reproduced the whole cloud at an angle of 68 degrees in the positive sense. This would turn it well to the left so that the same surface would be reproduced in the cloudlet at the part where it seemingly emerged from the main cloud.

The formulas for *a*, *b*, *c* and *d* are based on the theory of affine transformations which, luckily, needn't concern us here. As it happens, simple trigonometric functions do all the work:

$a = .214\cos(68)$ $b = -.214\sin(68)$

$c = .214\sin(68)$ $d = .214\cos(68)$

In each case, the contraction factor is multiplied by the cosine or sine of the angle of rotation. If we calculate these numbers on a hand calculator, the first four numbers in row B of the cloud's IFS table emerge.

The translations, *e* and *f*, turn out to be especially easy to calculate. The shrunken and rotated sub-object has the same relation to the origin as the whole object does. Where does the origin go when the sub-object is at last translated to its final resting place? The horizontal distance is *e* and the vertical distance is *f*.

By now, some readers will be pondering some elementary creations of their own. How to make a mountain or fire? Should a two-dimensional or one-dimensional collage be used? The answer to the second question depends on whether the fractal appearance aimed at is essentially one- or two-dimensional. Take mountains, for example. I can imagine a mountain in outline as a series of sub-mountains that mount to the summit. I can also imagine it as a more or less solid object with a fractal outline.

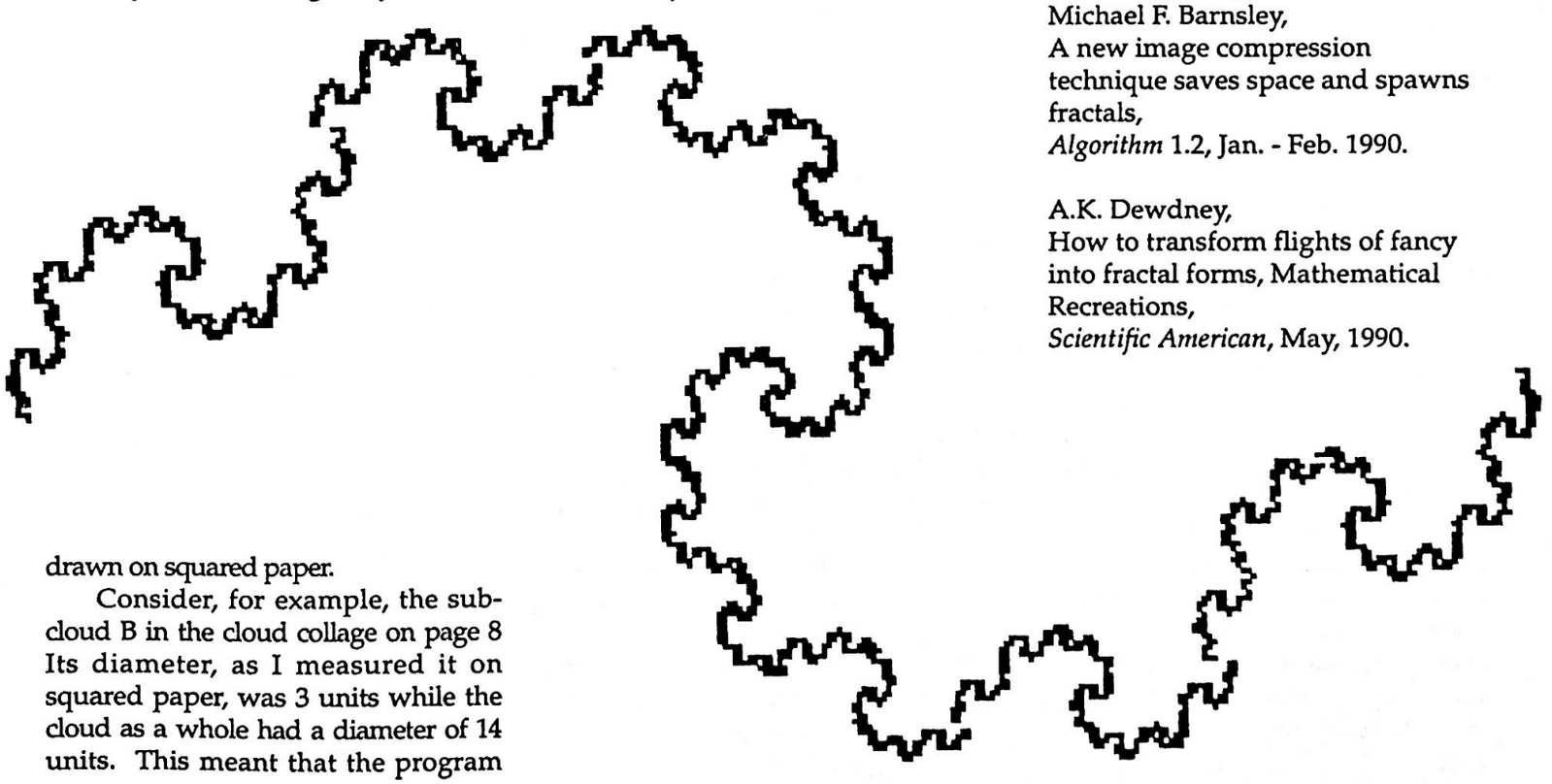
Only when I have received some mountain and fire pictures from readers will the four elements be complete. I will be happy to publish the best images sent to me:

*Dept. of Computer Science*
*University of Western Ontario*
*London, Ontario, CANADA*
*N6A 5B7*

***Further Reading:***

Michael F. Barnsley,
A new image compression technique saves space and spawns fractals,
*Algorithm* 1.2, Jan. - Feb. 1990.

A.K. Dewdney,
How to transform flights of fancy into fractal forms, Mathematical Recreations,
*Scientific American*, May, 1990.



# PERSONAL PROGRAMS

by Clifford A. Pickover

## *A random walk through a digital garden*

*"Mathematicians are inexorably drawn to nature, not just describing what is to be found there, but in creating echoes of natural laws."*
Roger Lewin

The use of "controlled accident" has found a place in modern art. The paintings of Louis Daguerre and Jackson Pollack and the music of John Cage are just a few examples. The greatest artist, mother nature, also exhibits intricacy and beauty in partially-ordered structures such as clouds, oceans, bloodvessels, mudcracks, minerals and trees. In studying certain geometrical aspects of random walks, I have run across several botanical, even artistic, effects. These can be created with the simplest of algorithms.

Random walks are sometimes portrayed colorfully as the path described by a drunk wandering aimlessly around on a sidewalk. If you were to trace the path described by a staggering drunk, you would not find it very interesting from a graphical point of view. But the study of the drunk's motion under various constraints can bring us to some useful and interesting general conclusions.

Consider, for example, the so-called one-dimensional random walk. Here, the drunk is confined to a tunnel in which he may go forward or back at a given step. There are probabilities, *p* and *q*, associated with each possible direction. At any given step, the drunk goes forward with probability *p* and backward with probability *q*. Mathematicians call these transition probabilities. If *p* is much greater than *q*, the drunk will have a pronounced tendency to go forward rather than backward. It is also possible to walk randomly according to transition probabilities that depend on the direction of motion. I will use this approach in constructing a garden of the strange-looking plants I call randophytes. Readers who wish to learn more about the subject can consult the article in *Mathematics Magazine* cited at the end of this column.

In the one-dimensional case, a direction-dependent random walk involves a change in direction with probability *p* and no change in direction with probability *q*. In other words, as the drunk staggers along the tunnel in a given direction, he will change direction with probability *p* and not change direction with probability *q*. Here, no matter what the relative values of *p* and *q*, there would seem to be no preference for the drunk to move in any particular direction in the long run (or walk). As a matter of terminology, we say that if the drunk changes direction he was "reflected," otherwise, he was "transmitted."

In the two-dimensional space in which my digital garden grows, I use random walks with transition probabilities that depend on the direction of motion of a particle in a polar space. By this I do not mean a frozen wasteland, but a two-dimensional space in which every point is specified by polar coordinates. Instead of distance north or west of given axes, polar coordinates use one distance and one angle. The distance is from the given point to a special reference point called the pole or origin. The angle is made by the line joining the two points with another, fixed reference line. Within such a coordinate framework, we may imagine a particle that wanders outward along radii that emanate from the origin, moving laterally in accordance with changes in the angle.

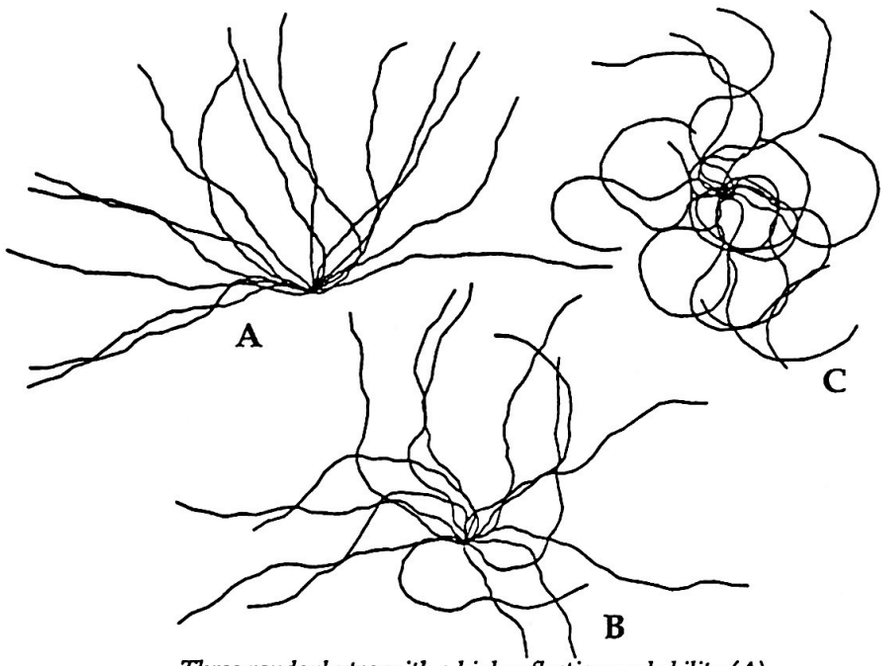
Spelled out in plain terms, imagine a particle that has polar coordinates *r* (distance from the origin) and *a* (the angle its line makes with the fixed line). The algorithm shown in the box below is based on the following scheme for

### Randophyte Algorithm

```
select initial position (x0, y0)
for j <— 1 to branchnum
      x <— x0, y <— y0
      move to (x,y)
      r <— 0, a <— 0
      num <— random
      if num > .5 then oldir <— 1
                  else oldir <— -1
      for i <— 1 to n
          if oldir = -1 then t <— p
                        else t <— 1-p
          num <— random
          if num  > t then dir <— 1
                        else dir <— -1
          num <— random
          num <— num x dir
          a <— a + num x da
          r <— r + dr
          x <— r x cos(a) + x,  y <— r x sin(a) + y
          draw to (x,y)
          oldir <— dir
```

list of key parameters and typical values:

| | | |
|---|---|---|
| branchnum: | number of branches | (20) |
| da: | increment for a | (0.5) |
| dr: | increment for r | (0.05) |
| p: | transition probability | (0.1) |
| n: | number of steps | (35) |
| x0,y0: | initial position | |




*Three randophytes with a high reflection probability (A), a medium probability (B), and a low one (C).*

reflecting or transmitting the particle as it wanders away from a particular starting point.

The motion of the particle will depend heavily on the choice of the parameter *p*. This probability drives the particle, so to speak.

1. At each step, the particle moves outward by a certain increment. In other words, its *r*-coordinate continually increases.
2. The algorithm keeps track of the particle's last angular motion with the variable *dir*. If *dir* was last positive, then a threshold *t* is set equal to *p*, otherwise, it is set equal to 1-*p*.
3. A random number is then selected and compared with the threshold. In this way, a new direction, *dir*, is chosen on the basis of the old direction and the probability *p* that the direction will be reflected. The particle is then moved outward by the increment *r* and laterally by a random amount in the direction *dir*.

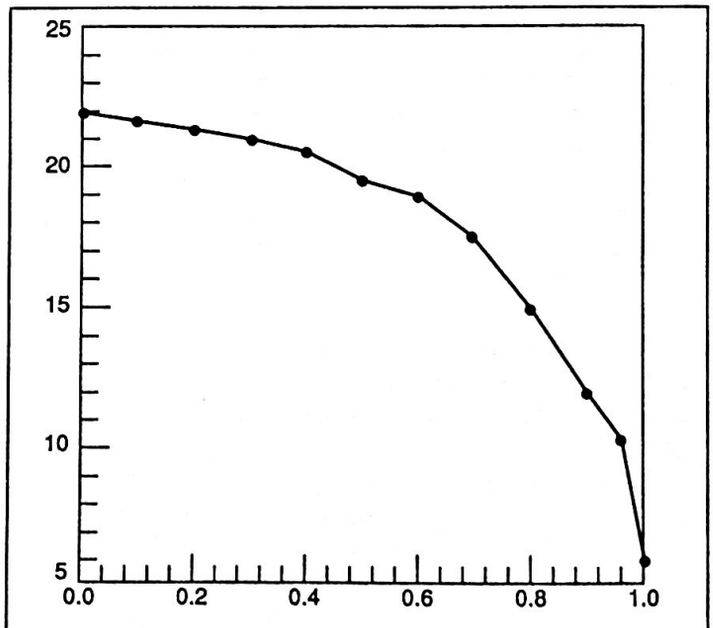
Steps 2 and 3 are repeated n times in the inner loop. The parameter *n* therefore controls the length of the random walk or, if you prefer, the length of the branch or tendril currently being "grown." It is interesting that small values of *p* will encourage lateral growth in a single direction with only occasional reversals. A value of 0.5 will result in a branch that creeps more or less in the outward direction. This is the normal walk of the staggering drunk in polar form.

The first time through the inner loop, the particle had no previous direction (*oldir*) so one has to be made up by selecting a random number, *num*, and using it as the flip of a coin. If the number exceeds .5, then *oldir* will be taken as 1, otherwise, as -1.

Other important parameters that control the randophyte are *branchnum*, the number of branches to be grown, *da*, the angular increment (in whichever direction), *p*, the transition probability, *n*, the number of steps in the branch and (*x*0,*y*0), the initial position of the particle. Typical values for these parameters are given at the bottom of the box on page 12.

Readers will notice that the algorithm only draws a cluster of tendrils from the given point, (*x*0,*y*0). One way to obtain more complicated plants is to start a new cluster at the end of each branch by making the algorithm a procedure and having it call itself at the end of every branch. In this way, one can easily obtain fractal-like structures that, in some cases, are rather reminiscent of real plants. I must admit, however, that the plants on page 14 have a distinct resemblance to certain kinds of seaweed.

The figure on the left shows the effect of *p* on some simple randophytes. One of these, plant A, has a *p*-value of 0.9. Consequently, its tendrils show little tendency to curl and they grow more or less outward. Plant B has a *p*-value of 0.5. Its tendrils show more of a



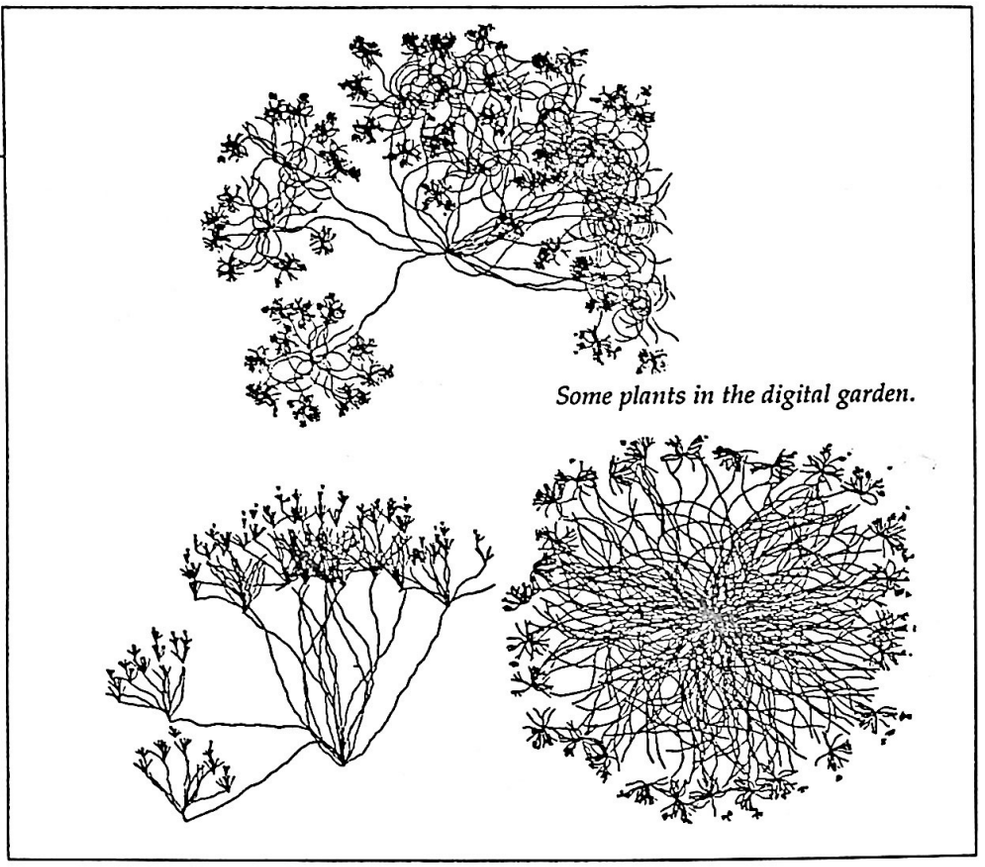

*How does tendril length change with transmissivity?*

tendency to meander without becoming as tangled as the tendrils of plant C, where the *p*-value is 0.1.

Growing randophytes is fun, of course, but some gardeners may be overcome, as I sometimes am, by the urge to experiment. Having noticed that randophytes with very low reflection probability have a strong tendency to occupy a smaller patch of ground, I began to wonder whether there was a statistical relationship between the distance a tendril gets from its starting point and the probability, *p*, of reflection. I carried out the basic growth algorithm just described many times with a varying *p*-value. Then I plotted the average distance the tendril got from its starting point as a function not of p but its inverse, 1-*p*, or transmissivity. As the latter increased from 0 to 1, I found that tendril length dropped along the rather regular-looking arc shown in the second figure on page 13.

The plant kingdom and the world of mathematics are generally perceived as having little in common. However, some plant forms very nicely illustrate certain classical problems in mathematics. For example, the famous Fibonacci numbers may be found as actual lengths along the stems of some plants. Similarly, developmental systems, in which a kind of mathematical plant is grown from a single cell by various developmental rules, are reflected in algae, a certain kind of wild cabbage, and other forms. In my book *Computers, Pattern, Chaos and Beauty* (see Further Reading), readers will find additional information on the subject of botanical patterns. Available by June of this year, the book also provides a host of graphic objects that have biological and artistic overtones.

In the meantime, there are gardens to be grown. As usual, I will be happy to see special productions by digital gardeners, especially those who discover new species.



*Some plants in the digital garden.*

## *ANNOUNCING A PATTERN BOOK*

I am now inviting readers to send me two-page manuscripts for a book entitled *The Pattern Book: Recipes for Beauty*. The book will contain hundreds of aesthetically and scientifically interesting patterns. Not all of them need be generated by a computer, however. Patterns that show some regularity or design, that delight or intrigue, are welcome. If there is one particular pattern that has intrigued you, why not send me the pattern on one page, along with a description of how it was produced, on another page?

To save time, it would be a good idea to write me at the address on the right for an "author's guide."

## *THE MAILBAG*

I have received several interesting letters from readers of past columns in *Algorithm*. Next issue I will give more of these, but for now, I should mention a discovery by Dr. Philip Barak of the University of Minnesota in St. Paul, Minn. After reading about Lorenz attractors in *Algorithm* 1.2, Barak carried out an extensive study of the ability of the Euler algorithm to solve various exotic differential equations. He and his Ecosystems class discovered that the Euler method does not always give a good solution to a differential equation–a warning to all!

*As always, I would be happy to hear from readers who develop the ideas I have described here. Please address your correspondence as follows, being sure to include a self-addressed envelope and stamp (if possible) if you would like a reply.*

*Clifford Pickover*
*Personal Programs*
*37 Yorkshire Lane*
*Yorktown Heights, NY 10598*

***Further Reading:***

L. Cohen,
Random walks with transition probabilities that depend on the direction of motion, *Mathematics Magazine*, vol. 37, no. 4, 1964.

C.A. Pickover,
*Computers, Pattern, Chaos and Beauty*, St. Martin's Press, New York, 1990. (available in June 1990)

# ALGOPUZZLE

by Dennis Shasha

## *Doctor Ecco turns up some bad pennies*

The telegram arrived at Ecco's MacDougall Street apartment on a rainy late autumn. He read it to me.

"Hmmm, from an Inspector Ross O'Reilly in Dublin: "DISPUTE OVER A WILL. PLEASE TAKE AER LINGUS FLIGHT 217 TO-NIGHT. WILL MEET YOU AT ARRIVAL GATE DUBLIN." I wonder if your students can spare you for a trip to Ireland."

Over the Atlantic, Ecco slept fitfully, sometimes humming a tuneless version of "Molly Malone."

At the Dublin arrivals gate, Inspector O'Reilly looked smart in his freshly pressed uniform, but his ready smile and engaging manner made him seem less a policeman than a pub-mate. Ignoring our grogginess after the night flight, he directed us to his unmarked car and proceeded to drive us to the Braycot mansion. On the way, he explained his reason for the urgent telegram.

"The Earl of Braycot," he explained, "earned the love of all Dublin. He was generous, witty, and a symbol of hope for many. You see, he was paralysed from the waist down, yet never complained, always finding ingenious new ways to overcome his disability. Some of his inventions, such as the Braycot arm-walker, have worldwide patents.

"His children, wouldn't you know, are an irresponsible lot. At Trinity College, here in Dublin, it was a rare month when some Braycot didn't wreck an 18th century dormitory room.

"Anyway, the Earl died of heart failure after a jolly dinner with his family last Wednesday. At least, he went out with a smile. To be sure, he was full of the Blarney and might even have the last laugh. He left a peculiar will that challenges his would-be heirs to a puzzle, in fact, similar to a puzzle he once gave to an international contest. I understand, gentlemen, that the two of you were winners in that contest."

I could feel Ecco's glance. He had been upset at my indiscretion in publishing the story of that contest in *The Adventures*. The only difficult puzzle in the contest had been:

*You are given 20 coins. Some are fake and some are real. If a coin is real it weighs between 11 and 11.1 grams. If it is fake, it weighs between 10.6 and 10.7 grams. You are allowed 15 weighings on a scale (not a balance). You are to determine which coins are real and which are fake.*

Ecco turned to me and asked, "Do you recall how we solved that other problem?"

"Yes," I replied, "we divided the 20 coins into 5 groups of 4 each and figured out which were the fakes in each group using only three weighings."

"Oh yes," said Ecco, "we were clever about that, weren't we. Take four coins, name them A, B, C and D. Weigh A, B, and C. If none or three are fakes, then simply weigh D and you're done. Otherwise weigh A and D. Then weigh B and D. If either pair contains both good or both fake, then it is easy to determine the quality of all coins. The remaining case is that each pair contains one good and one fake coin. In that case, if A,B, and C include two good coins, then D must be fake and A and B must be good; if A, B , and C include two fakes, then D must be good and A and B must be bad."

"YOU were clever," I replied. "For your own perverse reasons, you gave me the credit."

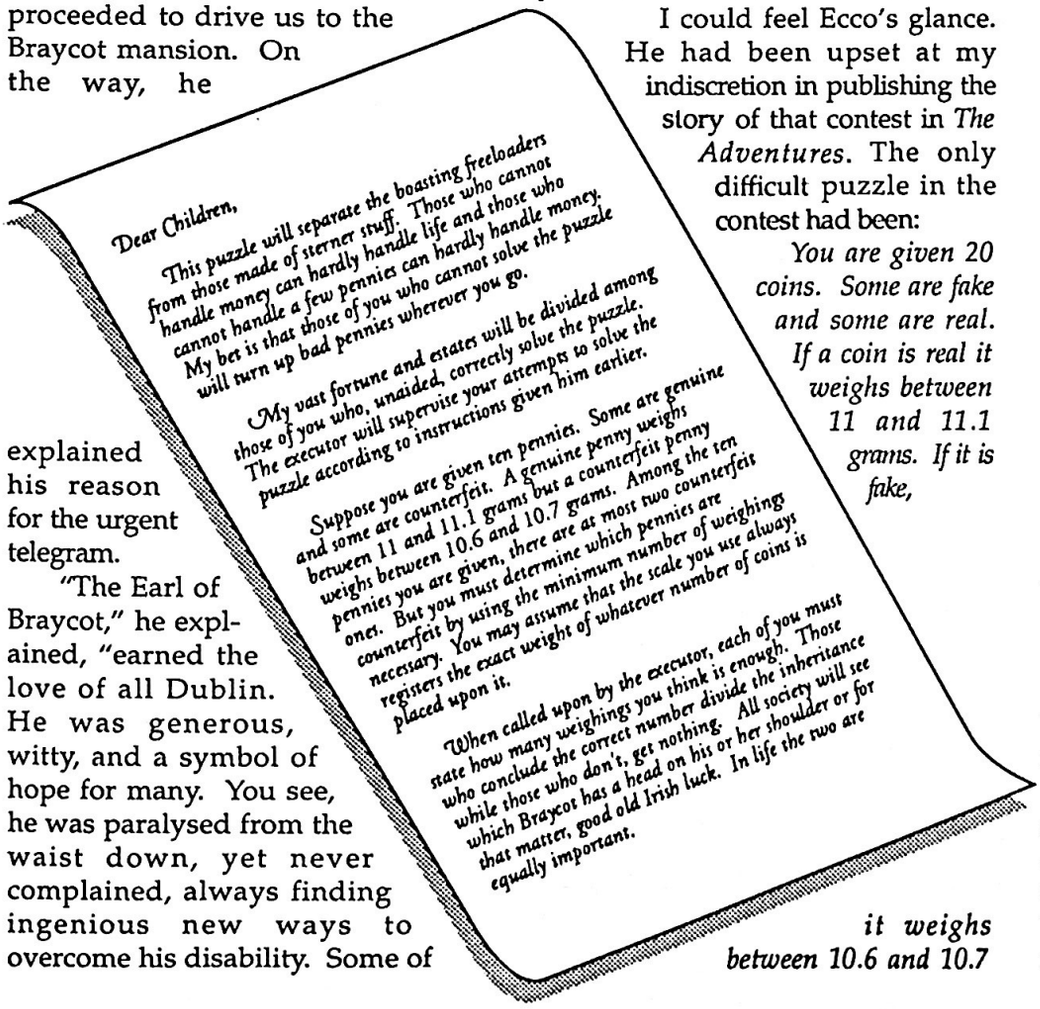
"Well, that's the one: the puzzle's from the Professor's book. The Earl proposed a variant of that puzzle."

When we arrived at the ancestral Braycot home, the Inspector ushered us in to a private dining room and slid a copy of the will across the table. The will (shown on the left) began as follows:

"Dear Children,

"This puzzle will separate the boasting freeloaders from those made of sterner stuff..."

The will recapped the basic setting of the problem, but it used 10 coins instead of 20, there being at most two fakes among the 10. When he saw that we had finished reading, the Inspector spoke up.


Dear Children,

This puzzle will separate the boasting freeloaders from those made of sterner stuff. Those who cannot handle money can hardly handle life and those who cannot handle a few pennies can hardly handle money. My bet is that those of you who cannot solve the puzzle will turn up bad pennies wherever you go.

My vast fortune and estates will be divided among those of you who, unaided, correctly solve the puzzle. The executor will supervise your attempts to solve the puzzle according to instructions given him earlier.

Suppose you are given ten pennies. Some are genuine and some are counterfeit. A genuine penny weighs between 11 and 11.1 grams but a counterfeit penny weighs between 10.6 and 10.7 grams. Among the ten pennies you are given, there are at most two counterfeit ones. But you must determine which pennies are counterfeit by using the minimum number of weighings necessary. You may assume that the scale you use always registers the exact weight of whatever number of coins is placed upon it.

When called upon by the executor, each of you must state how many weighings you think is enough. Those who conclude the correct number divide the inheritance while those who don't, get nothing. All society will see which Braycot has a head on his or her shoulder or for that matter, good old Irish luck. In life the two are equally important.

"Well, there are going to be some poor Braycots in this part of the world," he chuckled, "The Earl's ten children have guessed every number between 3 and 10, two guessed 7 and two guessed 6.

From a room nearby we could hear the noise of a loud and continuing argument.

"You can hear them having at it. Each of them is absolutely convinced of the rightness of his or her answer. They've only agreed to one thing — to hire an expert who can make a final ruling on who is right and who is wrong. I don't mind telling you that you can take a few years off with the fee that will come with this job!"

Ecco nodded in gratitude, then asked, "So, you want me to demonstrate to everybody's satisfaction that a certain number of weighings is enough, no matter whether zero, one or two pennies is a fake, then to show that fewer weighings are not enough, is that right?"

"Yes, sir, and quickly. It sounds like they might be presently coming to blows over the matter!"

The voices in the nearby room were rising to a crescendo. Ecco turned to me.

"Well it still looks as if we can only weigh three coins at a time to be sure of the number of real and fake coins in a weighing."

*Find the minimum number of weighings under the assumption that at most three coins can be weighed at once. Show that fewer is too few.*

Ecco made me wait outside. With exceptional courage, he entered the room with the Inspector and there revealed the true minimum number to gathered would-be heirs, at most two of whom were destined to earn the Earl's millions.

It occurred to me as I stood waiting for the resolution that true algopuzzlists might try to find algorithms that involve weighing more than four coins together at a time, even with the attendant ambiguity.

Solutions to the two coin-weighing algopuzzles will be published in *Algorithm* 1.5.

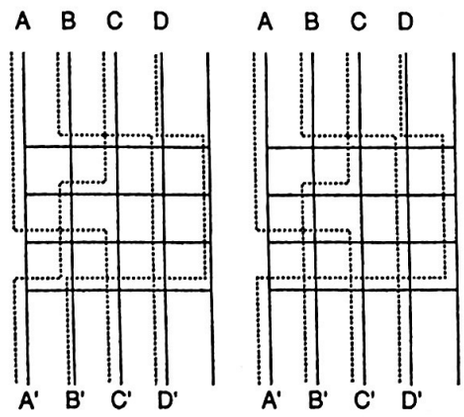
Here are the solutions to the three Algopuzzles in *Algorithm* 1.3:

There is no way to have a race of four bulls on four tracks where even one bull can cross a track and still have the routes meet only at the intersections. To see this suppose that the bull starting at X is to end at Y'. Clearly, the route that the bull starting at X takes must use a cross-track. Suppose the first cross-track it uses is cross-track $i$. But then on track X-X', between cross-track $i$ and $i$+1, no bull can run. Therefore, only three tracks are available between cross-track $i$ and cross-track $i$+1. This is not enough.

With five tracks, but only four bulls, four cross-tracks are enough. First, the permutation with at least one straight route (i.e., one that uses no cross-tracks) reduces the problem to one of three bulls on four tracks. This is handled easily. If there are no straight routes, then there are two fundamentally different possibilities: two exchanges (i.e., two permutations of two tracks each) or zero exchanges. Two representative hard cases are: A-C', B-D', C-A', D-B', and A-C', B-D', D-A', C-B'.

The permutation A-D', B-A', C-B', D-C' requires only two cross-tracks.

George Berkin of Yardley, Pennsylvania, Cornelius Groenewoud of Bartow, Florida, and Bill Mason of



Hornitos, California have closed another question of spy encoding from Algopuzzle 1.2. For any collection of fourteen 4-bit messages, does there exist a string of 18 or fewer bits that contains all the messages? Mason showed that 18 bits were enough for any fifteen(!) 4-bit messages. Berkin and Groenewoud showed that 17 bits were not enough for certain sets of fourteen 4-bit messages (e.g., all 4-bit messages except 2 and 3).

*Address all correspondence as follows:*

*Prof. Dennis Shasha, Algopuzzle*
*Courant Institute of*
*Mathematical Sciences*
*New York University*
*251 Mercer Street*
*New York, NY 10012 USA*

*Please enclose a self addressed (and, in the US, stamped) envelope if you would like a reply.*

**Further Reading:**

Dennis Shasha,
*The Puzzling Adventures of Dr. Ecco,*
New York, W.H. Freeman, 1988.

# ALGOFACT

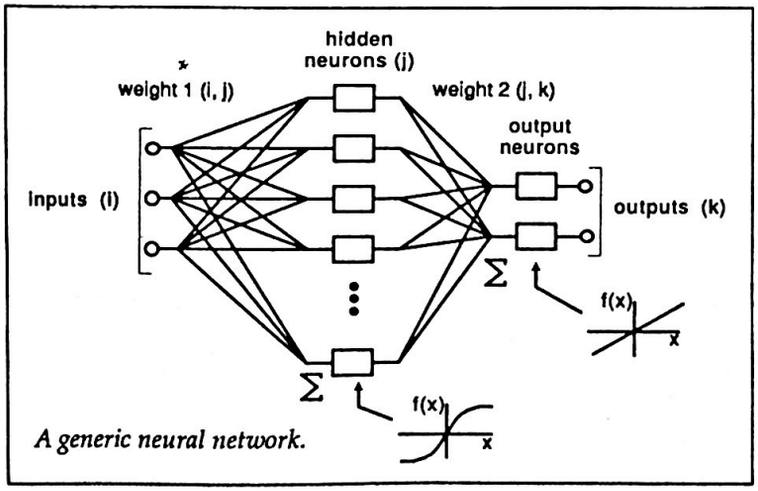
by Edward A. Rietman and Robert C. Frye

## *Neural networks that learn*

Neural networks are systems designed to mimic the learning function of the brain. Through trial and error, they gradually modify their own behavior until they produce acceptable results. And, like the brain, their ability to learn is limited only by their architecture and by feedback from outside sources during the learning process.

Here we will show you how to program a simple neural network on your personal computer. We will use the network to convert polar coordinates to Cartesian coordinates. Similar neural networks are able to "learn" most non-linear calculations to a high degree of accuracy.

The underlying physical models for neural networks have developed from early attempts to model the behavior of the brain. They generally consist of layers of active elements, the *neurons*, coupled together with variable-strength connection matrices that simulate the *synapses* found in the brain. The neurons in such networks can be thought of as algebraic functions applied to a weighted sum of all their inputs, and the synapses as the weighting factors that apply to the inputs. In the figure below, neurons are represented by rectangles and synaptic connections by the lines connecting each node.




*A generic neural network.*

Layers of neurons not directly connected to the outputs are often called *hidden layers*. A neural network can contain any number of hidden layers, but only one will be used in the polar-Cartesian conversion example.

Each neuron acts as both a summing node, collecting a weighted sum of all signals in the previous layer, and as an *activation function*, applying a response curve to the weighted sum. The hidden neurons are usually chosen to have a sigmoidal, or step-shaped, activation function as shown in the figure. The hyperbolic tangent function, which we will use in this example, and the arc tangent and Fermi functions are commonly used in neural network software, all of which have the correct shape and give comparable results. Their sigmoidal shape is important. The non-linearity can be used by the adaptive network to construct non-linear output responses. However, their saturating nature keeps the responses of the overall network nicely bounded, preventing uncontrolled behavior.

The activation function for the output neurons in the last layer, on the other hand, is usually linear. This gives them an unlimited dynamic range, allowing the network to generate arbitrarily large output signals.

Apart from the number of layers and the number of neurons in each layer, the basic architecture and elements within the network are generic. One network of this kind looks more or less like any other. The function of the network is not determined by its active elements, the neurons, but by the interconnections among them. The basic idea is to adjust the values of the individual connections, *weight1(i,j)* and *weight2(j,k)*, to minimize the error in the output. This process, the learning process, is gradual. The changes to the connections after each individual trial are small, but with each change, the network more closely approximates the desired response.

### *LEARNING FROM MISTAKES*

A widely used adaptive method of computing the required changes in the connection matrices is the *back-propagation of errors* technique (see Rumelhart's book, listed in Further Reading). It attempts to minimize the mean-squared error in the output of a network as compared to a desired response, or target. For a network having multiple outputs, the error is given by the square root of the following formula:

$$\sum_k (output(k) - target(k))^2,$$

denoted by E.

If the network is time-invariant, then its outputs will depend only on its inputs and the current values of the connection weight matrices. For a particular set of inputs, the degree of error is determined by the values of the weighting coefficients that connect the network. The method used in the adaptive procedure is to calculate the difference between the outputs and the targets and change the connections by an amount proportional to the gradient of the error in weight space. So, for a given weight matrix connecting, for example, the inputs to the hidden neurons, $\Delta weight\ (i,j)$, is proportional to

$$\frac{\partial E}{\partial\, weight1(i,j)}$$

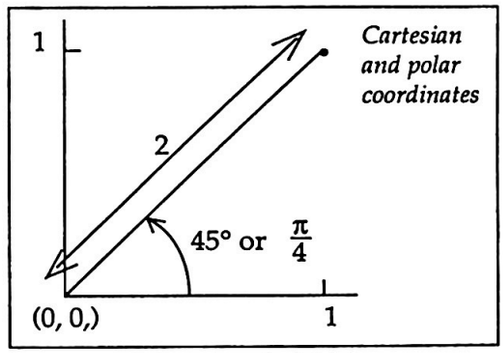
This technique results in lowering the average error as the weight matrices in the network evolve. In layered networks of the kind we are discussing, the changes to the weights after each trial are proportional to the error itself. This leads to a system that settles to a stable weight configuration as the errors become small. The changes become zero only if the gradient of the error in weight space is zero, which can represent either a true global minimum or only a local one. But from a practical standpoint, this gradient descent algorithm generally leads to useful solutions.

In a multi-layered network, the required weight changes are first computed for the last weight matrix, then for the next-to-last, and so on until the error correction has been propagated back through the network. Then another set of outputs is calculated for a new set of inputs, the outputs are compared with the target values, and the new calculated error correction is propagated back. Each time this process is repeated, the weight matrices will slowly adjust to generate output closer and closer to the targets.

## *PROGRAMMING A NEURAL NETWORK*

These general principles can be used to code a program that implements a neural network capable of learning through back-propagation of errors. We will use these principles to develop an algorithm that can "learn" to convert polar into Cartesian coordinates through trial and error.

Cartesian coordinates are the most common way of representing points on an $x$-$y$ plane. They are expressed as a pair of numbers $(x,y)$, representing distances along the $x$ and $y$ axes from the origin, (0,0). Polar coordinates are expressed as an absolute distance from a point to the origin, and an angle, in either degrees or radians, between the positive $x$-axis and the line connecting the point to the origin. For example, the point in the figure to the right could be expressed as (1,1) in Cartesian coordinates, or as $(\sqrt{2},45°)$ or $(\sqrt{2},\pi/4)$ in polar coordinates.




The variables used in the algorithm we will develop correspond to the labels in the diagram on page 17. In addition, *iunits* is the number of inputs, *hunits* is the number of hidden neurons, and *ounits* is the number of outputs.

The first step is to generate random values for all the elements in the weight matrices, *weight1* and *weight2*, by using a random function that returns a real number in the range -1 to 1. The box labelled Step One on page 21 displays this part of the overall algorithm.

This initial randomization of the weight matrices is important. Without it, the network starts out in a high symmetry state from which it will never emerge.

The next step is to select an input vector for the network, a pair of random polar coordinates. The angle can be expressed either in degrees or radians, depending on the requirements of the trigonometric functions built into your system. This example assumes radians. To use degrees, just change the constant ($\pi$) in the second line below to 180:

*input* (1) <– *random*
*input* (2) <– $\pi$ x *random*
*input* (3) <– 1.0

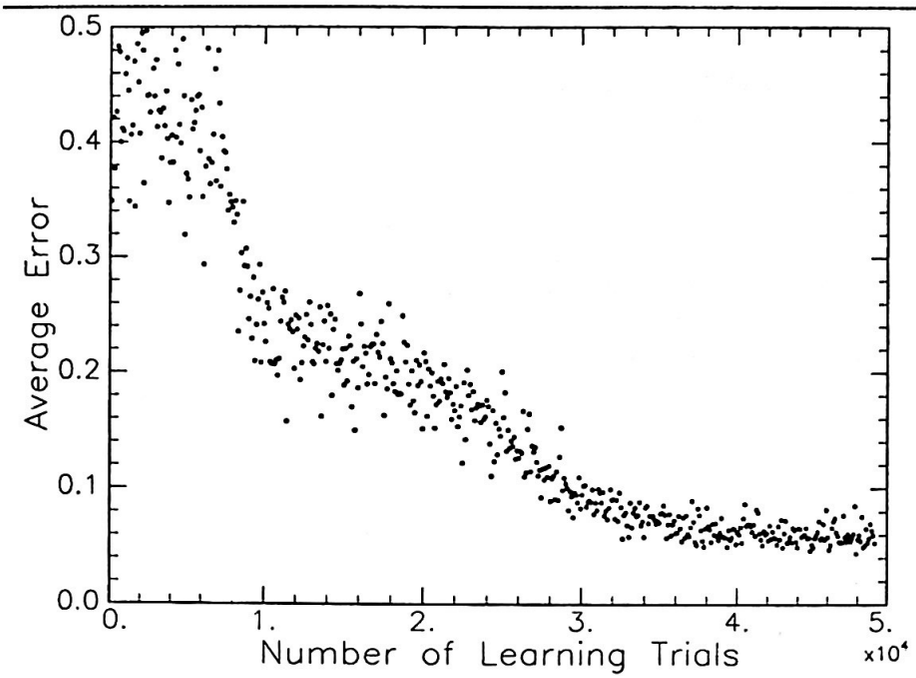
Note that in addition to providing the network with values for the radius and the angle, we have included a third input, the constant 1.0. This constant is important. Its value is not critical, but should be comparable to a value typical for the other inputs. Weighted values of this constant get summed into each hidden neuron along with the other inputs. This effectively gives the network an additional degree of freedom to shift the inputs by a constant, adjusting the threshold of the sigmoid functions in the hidden layers, and allows the network to generate non-zero outputs when all the other inputs are 0. For many learning tasks, the performance of the network can be poor without this constant.

Next, we calculate the target values, or desired results, for the conversion of the input values to Cartesian coordinates. The values will be used later to calculate the errors needed to change the weight matrices. This step is required only to teach the network to give acceptable results; it is not needed once the network has been trained.

*target*(1) <– *input*(1) x *cos*(*input*(2))
*target*(2) <– *input*(1) x *sin*(*input*(2))

The next step is to feed the inputs through the first layer of the network. Each input to the hidden layer, *hinput*(*j*), is the sum of the products of all the inputs multiplied by the corresponding weights. The output from each hidden neuron, *houtput*(*j*), is the result of applying a sigmoidal function, the hyperbolic tangent function in our example, to each hidden neuron input. The box called Step Two on page 21 summarizes this step.

Next, we feed these outputs from the hidden neurons forward to the output layer. This step is analogous to the previous step, but instead of applying a non-linear




*The learning curve of the polar converter.*

activation function, we calculate the error values—the difference between the network's output and the target response. These values will be used during back propagation to adjust the weight matrices. This step, (see Step Three on page 21), is needed only to train the network. It is not included once the network has been trained.

At this stage in the program, we have propagated the input values forward through the network, arriving at the network's output and its error. For a network that has been trained, i.e., one in which the weight matrices have been adaptively computed, that is all that is necessary to use the network. If the network is doing its job properly, the errors at this point will be small.

The following steps describe the procedure to use the error values to make changes to the two weight matrices. This procedure starts at the network's output, where we know the error, and computes the changes progressively backward through the network.

The first step in the weight update procedure is to use the error values to update the "hidden-to-output" weight matrix, namely *weight2*, as shown in the box labelled Step Four on page 21.

The *rate* is a constant that dictates how fast the network is to learn, or how much the weight matrices are to be adjusted each time a new set of input values is applied to these algorithms. If it is set too low, the network may take many trials to produce acceptable results. If set too high, the learning process will suddenly diverge and the matrix values will begin a catastrophic increase, resulting in an overflow. If the inputs and target outputs of the network are near unity, as in this example, a rate value of 0.1 works fairly well, but this may vary depending on network size. The best advice is to start with this value, and then adjust it as circumstances warrant.

The calculations for adjusting the "input-to-hidden" matrix, *weight1*, require two steps. We must first compute new values to be input into the hidden layer from the output layer (moving backward through the network) based on the adjusted weights in the "hidden-to-output" matrix, and then adjust the "input-to-hidden" matrix values accordingly.

In the first step, *sigma(j)* represents the values applied to the hidden neurons from the output layer, and *sigmoid(j)* represents the results of applying the inverse of the sigmoidal function to each of the values that were originally passed to the hidden neurons from the input layer.

In the second step, the "input-to-hidden" matrix values are adjusted so they will generate values closer to the desired inputs to the hidden neurons based on the previous calculations. Both steps appear as Steps Five and Six on page 21.

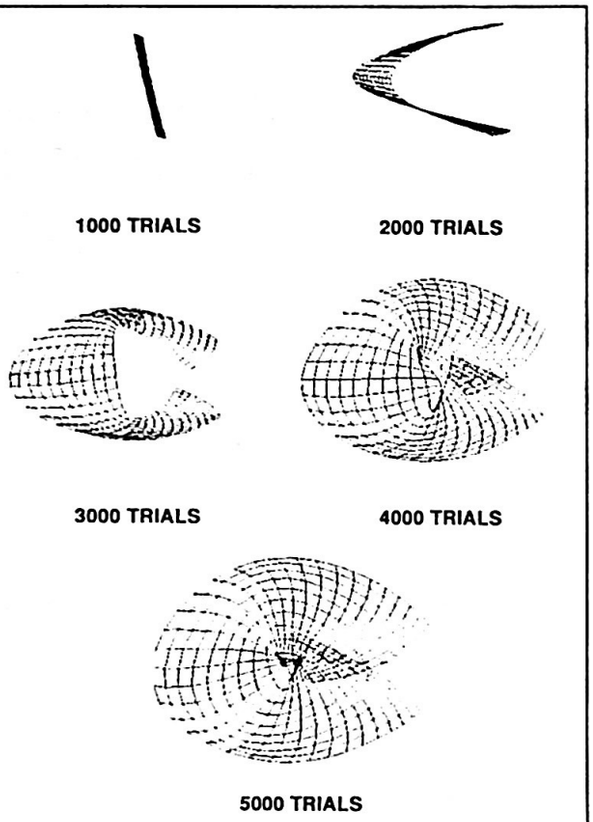
At this point, new random input values are selected, and the entire learning process is repeated until the minimum possible error rate is achieved.

## *MONITORING THE LEARNING PROCESS*

A way to monitor the progress of the network as it learns is to plot a running average of the magnitude of the error values as a function of the number of learning trials. After many trials, the error will level out, and the network will reach a point where further training is of little or no value. (Sometimes, however, you might observe plateaus in the learning curve that will drop off in subsequent trials.) If the error does not reach an acceptably low level, the network architecture needs to be changed. A word of caution, though: bigger is not necessarily better.

The learning curve on this page was plotted for a network having 40 hidden neurons in a single layer. The inputs to the network were constrained to lie inside the unit circle. In this network we used a learning rate of 0.01. The result was an error of about 6%.

Another way of monitoring the learning process is to occasionally




*The polar converter improves its accuracy.*

interrupt it and map the relationship that the network has learned. We can do this by setting the input radius to a constant value and allowing the angle to vary each time we want to observe the results. Ideally, the network's calculations for $(x,y)$ should generate a circle. Similarly, if we keep the angle constant and allow the radius to vary, the network should generate a line extending outward from the origin. If we apply both these tests, using several preset values for both the radius and the angle, we should see a series of concentric circles with regularly spaced "spokes" extending from their center. The figure above shows an example of such monitoring as the number of trials increases.

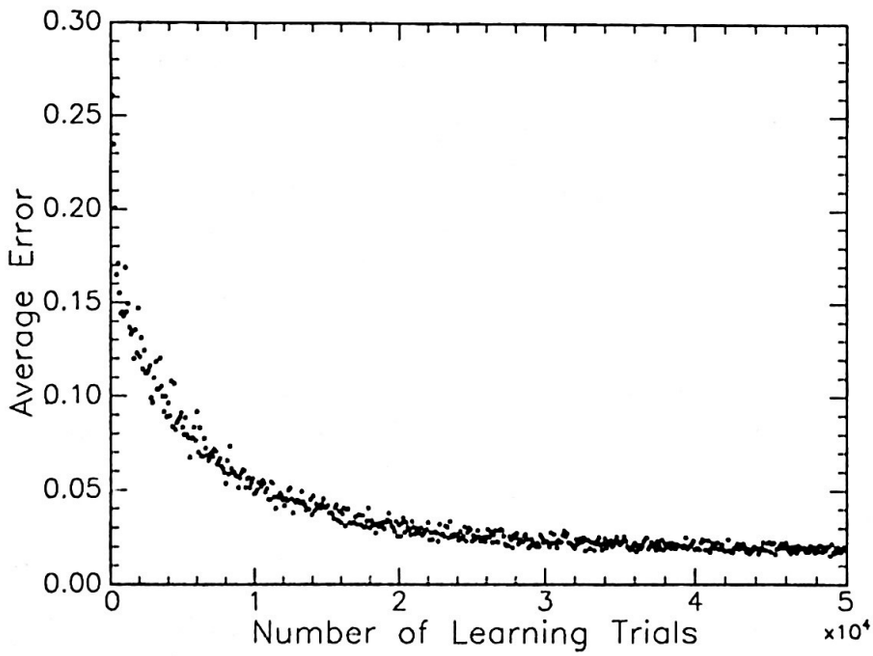
In the early stages, the elements in the weight matrices have not had a chance to grow large, so the output of all the neurons is relatively small. The sigmoidal neurons in the hidden layer are still operating in their small-signal, linear range, and the network's response is a small, rectangular grid. As the synaptic strength grows, the response becomes progressively less linear. As the number of learning trials increases, the network is able to produce a more accurate approximation of the ideal map. Beyond 5,000 trials, the accuracy of the network does not improve noticeably. With this particular choice of architecture, the network is not capable of doing better. The problem is particularly apparent near the origin, where the non-linearity becomes extreme and the network has some difficulty wrapping the map around the tight bends.

Changing the size of the neural network can have some surprising effects on the accuracy of the results. The graph below shows the learning curve of a network with two hidden layers of 15 neurons each. It achieved a performance level of about 1.5% error, even though the network has fewer neurons than in the previous example. Feeding one non-linear hidden layer into another one compounds the non-linearity. For some highly non-linear problems this can be a big advantage.

What happens if we increase the number of neurons in each of the two hidden layers to 150? Here, the network achieves about a 2% error, not quite as good as the smaller network. And nearly 80 times as many computations are needed to accomplish each trial as with the smaller network. Using a 286 machine with a 287 floating point chip, the small network computation time is a few hours, while the large one requires about one week for the same number of learning trials. Bigger networks do not necessarily result in better accuracy, but they always demand more computation time.

## *PRACTICAL APPLICATIONS*

The ability of adaptive neural networks to self-organize can be used to solve a variety of non-linear




***More neurons are better - sometimes.***

problems. The networks themselves are simple to program, and they can be used to emulate the responses of much more complicated systems. The resulting trained network, if successful, can be a more efficient way of obtaining equivalent results.

Another common use of these networks is to train them using experimental data from real-world systems that may be so complex that we cannot easily formulate equations or rules to describe them. Here we take advantage of the adaptive network's ability to extract a set of underlying rules from the training data. In many instances, the networks find rules that work better than those of a human programmer.

Unfortunately, we do not yet have a mathematical formalism for neural networks that allows us to say ahead of time whether they will work for a particular problem, and what the optimum architecture for the problem will be. That is still largely an experimental science. But with appropriate inputs and targets, the basic procedure described here can be used to set up and train almost any neural network.

*Edward Rietman and Robert Frye are researchers at AT&T Bell Laboratories in Murray Hill, N.J.*

**Step One**

```
for j <— 1 to hunits
    for i <— 1 to iunits
        weight1 (i,j) <— 0.1 * random ()

for k <— 1 to ounits
    for j <— 1 to hunits
        weight2 (j,k) <— 0.1 * random ()
```

**Step Two**

```
for j <— 1 to hunits
    hinput (j) <— 0
    for i <— 1 to iunits
        hinput (j) <— hinput (j) + weight1 (i,j) * input (i)
    houtput (j) <— tanh (hinput(j))
```

**Step Three**

```
for k <– 1 to ounits
    output (k) <– 0
    for j <– 1 to hunits
        output (k) <– output (k) + weight2 (j,k) * houtput (j)
    error (k) <– target (k) - output (k)
```

**Step Four**

```
for k <– 1 to ounits
    for j <– 1 to hunits
        weight2 (j,k) <– weight2 (j,k) + rate * houtput (j)* error (k)
```

**Step Five**

```
for j <– 1 to hunits
    sigma (j) <– 0
    for k <– 1 to ounits
        hinputrev (j) <– sigma (j) + error(k) * weight2 (j,k)
    sigmoid (j) <– sech(hinput (j))^2
```

**Step Six**

```
for i <– 1 to iunits
    for j <– 1 to hunits
        delta <– rate * sigmoid (j) * sigma (j) * input (i)
        weight1 (i,j) <– weight1 (i,j) + delta
```

**Further Reading:**

D.E. Rumelhart and J.L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Cambridge, MA, MIT Press, 1986.

E.A. Rietman, *Explorations in Parallel Processing*, Blue Ridge Summit, PA, Tab Books Inc., in press.

# SPECIAL REPORT

by William R. Buckley

## *Artificial Life in Santa Fe*

### *ARE COMPUTER VIRUSES ALIVE?*

After a hiatus lasting more than two years, the A-Lifers are back at it, this time in Santa Fe, New Mexico. Meeting at the Sweeney Center, during the week beginning February 5, more than 200 attendees of the five-day conference viewed the latest developments of their fledgling discipline. Major themes were the direction of A-Life research, topics for exploration, and ethics, such as professional responsibility and public disclosure. Details of current A-Life projects filled dozens of talks, posters, and video and computer demonstrations.

Major topics in A-Life include the origins of life, also called emergence; chaos, entropy and order; systems of self organization; and evolution. Among the highlights of the conference were studies related to cellular automata, hypercycles of ribozymes, computational models of individual and group behaviors, and artificial chemistries.

Artificial life might already be out there. A special conference panel took up the issue of computer viruses. Attendees Hyman Hartman (UC Berkeley), Russell Brand (Lawrence Livermore National Laboratory), Eugene Spafford (Purdue University), and Harold Thimbleby (Sterling University) were led by David Jefferson (UCLA Computer Science) through an analysis of the issue. With Hartman and Thimbleby for the defense, and Brand and Spafford on the offense, Jefferson provided the balance, acting as moderator.

Hyman Hartman introduced an interesting concept from biological genetics, that real viruses amount to a communication medium for new and useful genes between species. The idea is that species, being genetically, incommunicado, rely on virions to update cross-species gene pools. In summation, Hartman declared viruses "will be as big a revolution to the theory of evolution as Mendelian genetics has been. I just wanted to put in a good word for viruses."

Harold Thimbleby urged the application of computer viruses productively, much in the manner suggested by Hartman. Thimbleby envisions his Live Ware as a system of vectors for database update messages and program update conversions. In a current research effort, Thimbleby is developing communication media which actually spawn virus programs as part of regular operations. Thimbleby argues that his system provides plenty of controls, and that damage from unexpected Live Ware operations is unlikely.

Eugene Spafford presented an overview of computer viruses, what they are and how they work, all in the context of responsibility. Comparing virus writers to those who would poison water supplies. Spafford asked, "what do you think should be done to [that kind of] person?"

Russell Brand sought to impeach all manner of computer virus application, suggesting that "viruses, of any kind, are always a suboptimal mechanism for doing the job." Of military intelligence applications, their impact on the efficacy of Star Wars computer systems, and the transmission of genetic improvements, none found support from Brand. He argued persuasively against each.

Looking at the question of whether or not computer viruses are a form of artificial life, Jefferson gave a list of characteristics that would be expected to be present in artificial life forms. Among the nine characteristics proposed were: self replication, the expression of purposeful behavior, a metabolism (however primitive), and the ability to carry information. Only two characteristics were denied to computer viruses, namely, homeostasis (the ability to change with the environment), and group behavior (the ability to live in populations). Jefferson expressed his view quite clearly, suggesting that "computer viruses, primitive as they are today even, are the closest thing that we have worthy of the term Artificial Life so far."

In the most universal address given, conference co-organizer Christopher G. Langton (Los Alamos National Laboratory) talked of the environments in which living systems are found. Langton noted that living systems "ride on the edge of chaos." On one side of this edge are simple, predictable environments in which systems evolve to trivial steady state configurations. On the other side of the edge are unpredictable environments where chaos reigns, precluding the emergence of organized structure. Langton's argument is that living systems strive to remain at the transition zone between these extremes. Langton also hints that living systems may have mechanisms to influence their position within this topography.

In a side commentary on the field of A-life, Langton stressed the need for disclosure to, and participation of, the public in A-life activities. His argument is simple: how can the scientists of Los Alamos be trusted in a field which evokes the nightmares of Mary Shelley without a well-defined system of public disclosure and input? If A-lifers are not to be

considered as modern Frankensteins, such a system must be put in place.

Echoing Langton's words about the edge of chaos were systems of cellular automata. In one demonstration, a series of views showed how automata behavior became more self-organizing as the degree of chaos was increased. Organization grew, with more interesting local behavior appearing in the environment, until a critical point was reached in the chaos measure. The model then showed how an excess in the chaos measure precludes a further ordering of the system. As the chaos measure was again increased, all order was lost, and the system began to exhibit disorganization, as indicated by the measure.

Many A-life systems codify characteristics of extant biological life forms. For instance, the stages of the life of some plants, their development and growth, can be modeled by L-systems. Graphic images created by interpretation of particular L-systems show plant structures that are convincingly real. One note of sadness echoed through the halls of Sweeney Center. The inventor of L-systems, Aristid Lindenmayer, had recently succumbed to a grave illness.

David Jefferson, in a presentation of his own, urged that attention be given to the unanswered questions of biology. It is here that A-Life might shed light. Jefferson contended that A-Life work should seek not just the creation of new life forms, but also the explanation of existing life forms. He cited the questions of speciation--the why of the species--and sexual segregation--the why of the sexes. Both of these questions are intractable with current physical, chemical, and biological theories.

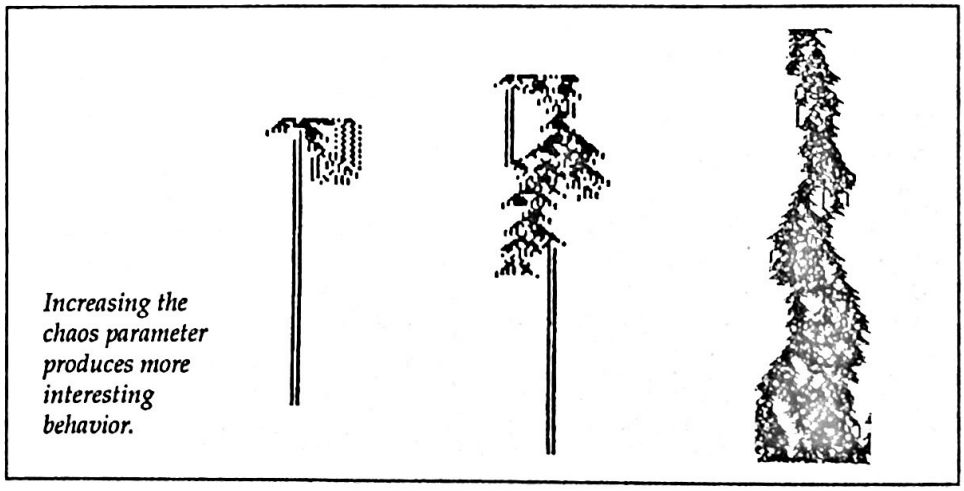
Gerald F. Joyce, with the Research Institute of Scripps Clinic, demonstrated what is for biologists the ultimate in A-Life research. Joyce is interested in the so-called "RNA world." In the early 1970s, Miller and Urey showed how a mixture of simple organic compounds could interact in an electrostatic environment to yield amino acid and nucleic acid monomers, and some polymers. Following up this work, Joyce expects to observe the evolution of higher order polynucleotides from a well-selected set of nucleic monomers.

By constructing high quality "wetware" environments composed of a restricted set of reagent grade molecules, Joyce has employed the recently discovered enzymatic character of certain RNA molecules to control the genetic elements of his system. The system demonstrates what seems to be a tangible argument for Lamarckian evolution, wherein two species co-evolve. Newly devised genetic codes can be incorporated into an existing genetic system. For those readers with a biological foundation, the star of the show was the Tetrahymena ribozyme which Joyce succeeded in getting to self replicate in a carefully controlled primordial soup.

In keeping with Langton's theme of professional responsibility and public disclosure, residents of the greater Santa Fe area were hosted to a show on the fourth night of the conference. Some of the projects more comprehensible to laypersons were given over to show-and-tell presentations. These included child-education-oriented systems, abstract computational art forms, poetry and a speculative inter-pretation of sulfur-based life forms.

A special attraction lay in store, presented by a crew from the Survival Research Laboratories: the famed Sonic Shock Cannon and its designer, Mark Pauline. A monster of superhuman proportions, the cannon is built from a forklift, fuel lines and tank, a combustion chamber eight inches in diameter and some fifteen feet long, and a supporting stand common to field artillery. The cannon fires a mixture of acetylene and oxygen, yielding a puff of wind travelling over two hundred miles per hour. As Pauline put it, "the puff strikes you like a pillow that is thrown at you." Set off in the main hall of Sweeney Center, the sound was unbearable, reverberating between the walls. The SRL crew obliged us with no fewer than five soundings of their tremendous beast.

A-Life is gaining in peer support, and those researchers who have not yet had the privilege of attending should mark their 1992 calendar for the next conference. Personally, I can hardly wait.



*Increasing the chaos parameter produces more interesting behavior.*

# ALGOFICTION

by Tekind Weywadeen

*The Arabian Nights is no match for a recursive tale.*

## *The Prisoner of Baghdad*

Ali Abu Riyah bounced thrice when the helmeted guard threw him into the deepest cell of the Sultan's dungeon. The wiliest rascal in all of Baghdad sat in the darkness wherein an ancient prisoner already dwelt among the rats and mouldy straw. When he recovered his faculties, Abu Riyah cried out, "Who is here?"

"'Tis I, Bisham" croaked the ancient prisoner, "Who art thou?"

"I am Ali Abu Riyah, Prince of Vagabonds, Master of Sages and Stealer from Thieves. I am committed to this darkness for but glancing at the Sultan's daughter."

"That is the way of the Sultan," said the old one, "He follows not the Law but the whims of his own evil nature. I have been confined in this darkness seventeen years for a crime I did not commit. But tell me your sentence. Perhaps I will have company for the next seventeen."

Abu Riyah shivered at these words, then he told his tale.

"It was but a small thing. Passing by the royal gardens, methought I heard such pleasant laughter as belongs in paradise and saw by the fountain such a one as the nightingales celebrate. Her hair was as the night which frames the moon - her face. And the form beneath her robes was as a gazelle that plays in the light of the moon. So much so, my breath disappeared and I swooned. My legs misgave me and I fell from the wall to the foot of a royal guard who took me to the Vizier."

"Ah, the Vizier! He telleth all the wicked punishments to the Sultan, who hath not the wit to invent them. But, tell me thy sentence."

"Passing strange it was. I have yet to think upon it. The Sultan said that I am to be executed at the dawn of one day this week. Perhaps it will be tomorrow, perhaps the next day, and so for five days upon that."

"How like the beast!" exclaimed the old prisoner, "He hath played with me in like wise. But, tell me, hath he not laid on strange and strenuous conditions?"

"That indeed he hath," cried Abu Riyah, now turning himself in the darkness to smile invisibly at his friend in misery "The Sultan allows but one ray of mercy to illuminate this dark prospect. If I correctly predict the day of execution, I will be set free, instead."

"By Allah," said the ancient one, "To give thee this tiny mercy he but doubles thy anguish. And how wilt thou determine the morn? I know not which case is the crueller, thine or mine:

"When I was but twenty, my friend and I were passing through the bazaar when a merchant ran from his stall and cried, 'stop, thieves! Those men have stolen my gold!' We were seized and taken before the Sultan. 'We have not taken this man's gold,' we cried, 'Search us as thou wilt. We are pious men, innocent of any wrong.' But the Sultan, who fancies himself very wise and the wit of the age, put on the face of Solomon and pronounced sentence. I remember the very words, as if he spoke them but yesterday:

'The prisoners are to be separated and confined in the dungeon. Each day they will be questioned as to their guilt or innocence. For as long as they maintain their innocence, they will be confined. But if, on the same day, both confess, they will have each a hand cut off and be released. Now, if one confess and the other not, the one will be set free and the other executed.'

"Each day, my gaoler comes to bring food and he asks. 'Hath thou somewhat to confess?' I say, 'By the beard of the Prophet, I am innocent.' My friend saith the same, for I have yet to lose my head."

"Thy friend is a true one," sighed Abu Riyah, "Every day he is sorely tempted to confess yet he holdeth fast to his innocence."

"And I, too. If I could but arrange a day when we both shall confess, that would be better than rotting here 'though it standeth against my principles."

"Nay. Thy principles are better. Mayhap Abu Riyah will find a way out for thee."

The ancient prisoner was much heartened by these bold words, yet, in his heart he doubted them. Who was this newcomer to brag so? And how could he help when, on some dread morn of the coming week, he would say to the gaoler, 'Today is the day of my execution' and it would not be so. Worse, he would say, 'Thou comest not to execute me this day,' and he would be executed! So thinking, the old one made a sigh to rend a heart of wood or to set the stones a-weeping.

"Thy heart rulest thy mind, old one. My mind engageth in plans, not morbid fears. By this discipline will I be released, and thee with me."

Abu Riyah then questioned the old prisoner most closely on the Sultan.

"Truly, he is the Fool of the Age. He hath barely the wit of a bird. Though he be a fool, all the greater for his pretensions to wisdom, yet he is a man of his word. When he setteth a condition, he will not be swayed from it, though his pride be wounded or his very life endangered."

Abu Riyah's eyes gleamed like those of a Djinn in the darkness, for

he had thought of a plan.

"Ha! He is a fool, this Sultan. He cannot execute me at all!"

The old man was astounded.

"Suppose there were but one day in a week. Then I should know the day of my departure from this life. When the gaoler came, I would say, 'today is the day of my death' and he would reply, 'Just so, Abu Riyah, thou art released by the Sultan's condition.'"

"But the week hath seven days, not one!"

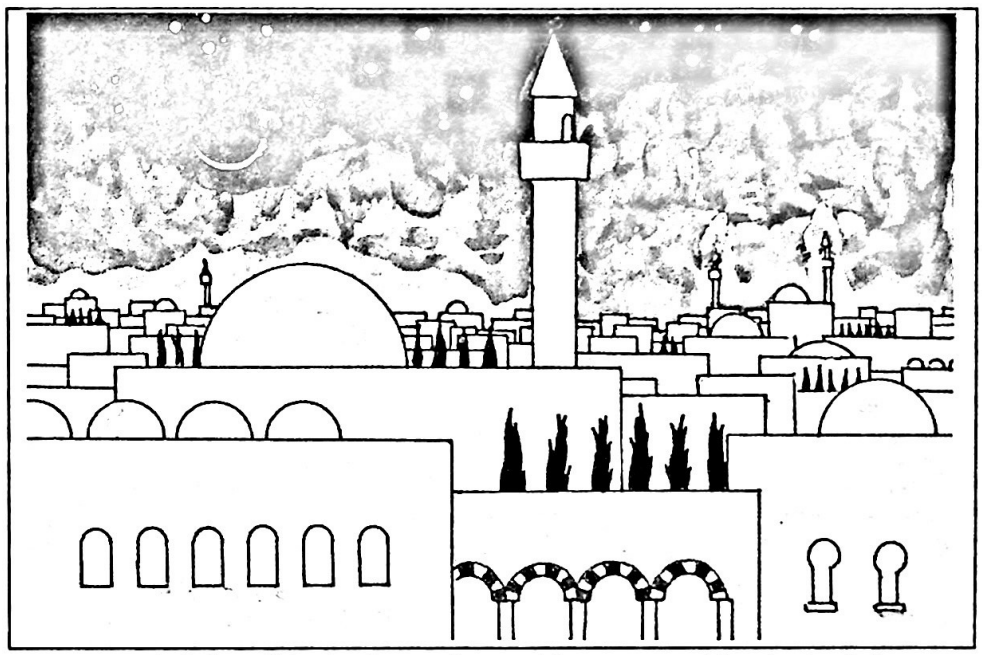
"Follow my caravan of thought: if the week had but two days, the second day could not be the day of my execution for I should know it. But that leaveth the first day as remaining and I should know that, too. Thus and so. Neither the first day nor the second could be my day of reckoning. Am I not already free?"

The old man bethought himself of Abu Riyah's argument. In a span of three days, four, or even seven, the same result would come: there was no day the Sultan could name as the day of Abú Riyah's death.

"I am tired now," said the Prince of Vagabonds, "I would sleep for I have much to do tomorrow when I am released from this pit of hell. I swear by the bow of the Prophet that by sundown tomorrow, thou too shall taste freedom."

So saying, Abu Riyah wrapped his cloak about him and brushed aside the rats to sleep upon the straw. But the old man stayed awake, thinking into the night. He bethought him of the bold words by which the Instructor of Sages and Stealer from Thieves might free himself on the morrow. Then he was minded how the Sultan might miss the argument and insist on his condition. With fear in his heart he reached out his bony hand to stir Abu Riyah awake.

"What if the Sultan deny thy argument? What if he hath not the wit to understand but presses on with his condition?"



Abu Riyah groaned. With much indignation he stood up and paced the cell a while. Then he went to the dungeon door.

"Gaoler! Gaoler!" he called.

The gaoler came, bearing a torch and a chain of rattling keys. Abu Riyah withdrew a fine pen from his cloak.

"See here a fine pen made from the horn of a unicorn and the feather of the great roc. Bring me some ink and seven sheets of paper that I might write my will thereon. When I have done writing, I will give thee this pen."

Presently the gaoler returned. With drawn sword, he entered the cell and left both ink and paper on the floor. The rushlight he let burn on the wall that Abu Riyah might see thereby. He glanced lovingly at the pen as he passed from the cell and locked again the dungeon.

In the light, Abu Riyah looked at the old man and saw one who should now be enjoying grandchildren about his knees and not rats about his feet. In his turn, the ancient prisoner looked wonderingly at Abu Riyah, a man of scarce thirty years with a look of the most piercing intelligence, a face quite unlike the Sultan's. He looked on in wonderment as Abu Riyah wrote and wrote. The pen made a scratching noise which drew all the rats. They crowded about the sheets of paper, as though to read what the Instructor of Sages might be writing.

The old prisoner fell asleep to dream of his friend, steadfast through the years, and to dream of his tiny children who were, Allah willing, now married with children of their own.

At the dawn of the first day, the gaoler came to the dungeon door. The torch was cold and the cry of the muezzin had long since faded with the dark. Here lay the old man and the young one, fast asleep.

"Awaken to the new day. What say you, Abu Riyah? Will you be executed this day? And what say you, Old Bisham? Hast thou somewhat to confess?"

Abu Riyah bestirred himself and rubbed his eyes.

"In truth, the Sultan cannot execute me at all. For not a day cometh but I might not know it!"

The words were braver than he felt for Abu Riyah no longer had confidence that the Sultan might

understand his mind.

"I plead my innocence!" cried the old man, with as much feeling as if he were asked for the first time.

"Then you must both come with me. The Sultan hath commanded that Abu Riyah be executed this day. Furthermore, this morning thy friend hath confessed, old man, and today is the day of thy death, also."

The old man wept a bitter tear but stood slowly up on tottering legs.

"By Allah, I am ready to meet my maker. Better to die at once than to live with the memory of a friend's treachery!"

So saying, he and the Spyer of Sultans' Daughters were led to the courtyard where the executioner stood waiting by the block, a great scimitar in his hand. Above him, in a gallery, sat the Sultan and his court. They were merry with much joking and jostling.

But Abu Riyah seemed calm and unmoved. He nudged the ancient prisoner and bade him be cheerful.

"Today we shall sip sharabs in the shade of a Tamarind," he laughed.

An official of the court read the charges and gave each prisoner the customary moment to speak his last words to human ears. First the old man spoke:

"I am innocent of any wrong. As Allah is my witness, I am wrongfully accused and wrongfully punished. I pray that the life to come is better than this one, albeit, my friend here hath promised that today we shall sip sharabs in the shade of a Tamarind."

The court gasped at such boldness. The Vizier stood up to speechify somewhat. He said that no Tamarind on Earth would shade their drink that day. Nor would it be in Heaven, being that the two were notorious as a thief and an adulterer. So saying, he sat down that Abu Riyah might have his say and so that the gallery might the sooner watch them pass through the portal of existence.

"I am but a poor magician and story-teller who, for one brief moment, gazed upon such beauty as would bedazzle any mortal. Albeit she is the brightest star of the Sultan's night and stands at the zenith of his sky, whereas I am at the nadir. Wicked I am and, contrary to what my friend saith, I expect shortly to stand before the angel of death and to say, 'Our Sultan is not to blame! Forgive him his fatherly zeal.' By this means my own sins, many as they are, might be forgiven."

The court was silent. Never had they heard such an eloquent speech from a dying man. At this very moment Abu Riyah drew a sheaf of seven papers from his shirt.

"I have here a magic tale which I would give to the Sultan as a parting present in light of all his excellent qualities. My punishment must not be delayed overlong, but let him agree that the story on these pages stay my execution for as long as it pleaseth him to hear it."

The Sultan was made happy by these words for he loved entertainment even more than compliments. His court would applaud his foresight in executing a man who would first entertain them. He waved his hand with a regal wave and Abu Riyah began to read from the pages in a loud voice, full of the most wonderful expression:

"This is the story of my Sultan, ruler of the age and wonder of our people. When still a child he mastered all the arts and sciences. When but a youth, he defeated whole armies in battle and made female hearts swoon with desire. As a young man, there seemed no matter beyond his grasp and no endeavor he had not mastered. Yet one thing remained, a secret he had not penetrated, a thing left undone, a great matter that escapes the knowledge of ordinary mortals."

The Sultan leaned forward in his seat.

"From an old sage he learned of the Hajradusat, a crystal kept in a secret cave. Whoever possessed the Hajradusat would possess infinite power, insuperable wisdom and life eternal. All the talents, riches, armies, harem and court of the Sultan faded beside the irresistible attractions of the Hajradusat. He was determined to have it.

So one day he betook him from the palace disguised as a beggar. The cave was in a far land and he durst not travel with pomp and army, for brute force would not win this ultimate prize, only great subtlety. For, just as the thorn guards the rose, so dark and malevolent forces protected the Hajradusat from all unworthy claimants.

The Sultan travelled thus in humble guise, sometimes kicked by donkeys and spat upon by ordinary people. Sometimes, indeed, old ladies would empty the contents of their chamber pots upon the Sultan from rooftops. Sometimes he was given foul leavings to eat and dungheaps to sleep upon. But withal he remained steadfast in his purpose. He said to himself when others laughed at his rude appearance, "You may laugh now, but I shall have the last laugh."

Presently he came to a land spoken of by the sage. Here the Sultan found a wondrous garden wherein was every kind of fruit and flower. Pomegranates and orange trees grew beside an endless stream that circled about the garden. Their branches were alive with every kind of bright-colored bird.

In the midst of the garden, in a brightly draped pavilion, lay a maiden of such beauty that my Sultan was sore afflicted with great

emotion and longing. With chaste and seemly deportment he approached the maiden. When he had come close to her, he saw that she wept copious tears.

He inquired most tenderly, "Oh Beauty of the Day, why dost thou cry?"

She replied, "I am prisoner here to the Djinn of the Hajradusat."

My Sultan asked, "What is this state and how came thee to it?"

The maiden replied, "The garden hereabouts is enchanted. If I but walk to its border I become forgetful of my purpose and wander back to this seat. I must stay herein until the Wonder of the Age shall come to rescue me."

"And who might be the Wonder of the Age?" asked the Sultan.

The maid replied that she knew not. Then, bidding the Sultan seat himself to eat of magic fruits and sweetmeats, she told him her story:

"I was but a tiny child when my father, an honorable man, was cast in prison by the Sultan for stealing what he had stolen not. My mother and sisters were without means to live and so we travelled to a neighboring town where dwelt our relations. But on the way we became weary and lay ourselves down to rest in the heat of noon. It was under the tree of a Djinn but we knew it not.

We were descended upon, of a sudden, by a black shadow that took us away in a swoon. I awakened alone in a great cave wherein seven doors were set. Towering over me was an evil Ifreet, a Djinn of terrible aspect, with skin like the bark of a tree and eyes like glowing coals. He said, "I am the Djinn of the Hajradusat. Thou wilt stay with me until thou art a maiden and suitable for bait. For I would go a-fishing with thee.' And here he laughed a terrible laugh, saying that only the one called the Wonder of Age might save me."

Our Sultan's heart, though covered with rags, beat the more quickly to think that he was that one. The maiden continued:

"The cave is in yon mountain. I go there in one day out of seven to serve the Djinn. In the cave, I have seen seekers after the Hajradusat come but never leave. The Djinn telleth them that the Great Crystal lieth behind one of the seven doors. If they choose the correct one, they are permitted to leave with the Essence of Knowledge. But if they choose the wrong door, great evil befalleth them."

"And what lieth behind the other doors, oh unfortunate maiden?" The Sultan of Beggars could scarce contain himself in the asking.

"It is a riddle that you may solve by hearing my story. Behind the first door lieth a pair of Lions that have not eaten for twelve moons. Behind the second door lieth two snakes that poison their victims. Behind the third door are two wasps the size of owls. Behind the fourth door are two armed warriors who will kill any man who opens it. Behind the fifth door are two dragons that teareth the flesh from any living thing.

"Concerning the sixth door, there came once a great Sultan to the cave in search of the Hajradusat. He came with his servant and I saw not which door the Sultan tried, for I was bidden to amuse the servant in the garden the while. As we sat by the endless stream, the servant told me of his master's quest:"

At this pass, Abu Riyah looked boldly up at the Sultan in the gallery and said, "And so the story continueth, my lord."

"Read on," cried the Sultan, for he was much excited by the tale.

"My throat hath become parched, oh, Light of the World," said the wily Abu Riyah, "Mayhap thy keen eye will read it better and faster and so draw this business to a more rapid conclusion. And when thou hast finished a page, but place it beneath the pages thou art reading."

So saying, he handed the magic story to the Sultan who took it eagerly and began silently to read the servant's tale: "This is the story of my Sultan ..." . The sun of noon came and went. The Sultan read on. Every now and then he exclaimed aloud, "By Allah, this tale groweth better and deeper as I read." Then, after reading through the afternoon, even while the executioner's blade was poised over Abu Riyah's head, the Sultan paused.

"This tale hath a marvellous subtlety that only a mind such as mine may enjoy. Thou art all dismissed, that I might finish it."

Abu Riyah and the elder Bisham walked from the prison, free men. They found their way to the Sultan's garden wherein the object of Ali's affection sat beneath the Tamarinds, sighing for the face she had seen over the wall but yesterday. When she spied Abu Riyah, her eyes outshone the stars now rising in the eastern sky. She gave them delicious sharabs to drink and the old man marvelled that Abu Riyah's prophecy had come true.

When Bisham could stand it no longer, he turned to the Prince of Vagabonds and asked behind which door the Hajradusat lay. Abu Riyah replied, "It mattereth not. The Hajradusat is not more valuable than life itself when each day is a door to death. Mayhap the Sultan will soon leave Baghdad in the guise of a beggar whereupon I shall return to the side of this beloved one."

"Mayhap he shall stop reading under advice of the Vizier," said the old man.

"If thou art correct, it behooves thee and me to travel far this night."

So saying, the Wonder of the Age said his salaams and disappeared over the garden wall.

# BULLETIN BOARD

*Software, games and publications from individuals and companies.*

### Computers in Science and Art

A catalog of Videotapes, Books, Software and more on Fractals, Chaos, Supercomputing, Cellular Automata, Computer Art-Animation, and Scientific Visualization. Created by leading researchers, for the professional, student and enthusiast. Dozens of select new products.

Write for a *Free* Catalog!

**Media Magic**
**P.O. Box 5079, Nicasio, CA 94946**

## PUZZLES, GAMES, BOOKS

- **BIPODS-** ***magnetic dissection puzzle with "pod control"*** **$8.00**
- **HEXPUZZLE -** ***very difficult*** **$15.00**
- **ELECTRIHEDRON -** ***hardwood*** **$35.00**

POST PAID
IN USA

**GENERAL SYMMETRICS, INC.**
2935 W. Chenango
Englewood, CO 80110

### POLY

Creates platonic and archimedian solids,prisms, antiprisms and their duals.
Rotations are fast with full hidden line removal on most figures.

Turbo Pascal Source Code Provided

MSDOS with EGA, CGA Graphics supplied on 5 1/4" 360K disk.

Send $15.00 to: **POLY**
**P.O. Box 983 Woden**
**ACT, Australia 2606**

### Expert Systems on your PC

IBIS by
Intelligence Manufacturing, Co.
Inexpensive, Easy, and Powerful
Educational Version-Shareware .....$40.00
Development System .....$250.00

Contact: **D.E.M. Software**
**P.O.Box 456 Woodbine NJ**
**08270 USA (609) 861-5181**

## AMYGDALA

A Newsletter of fractals & $\mathcal{M}$
Box 219, San Cristobal, NM 87564
Subscription, 6 issues, @ $15.00
Color Slide Supplement,
6 sets, 4 slides each, @ $19.00

### MATH PROGRAMS

THESE PROGRAMS DO HIGH SCHOOL & COLLEGE MATHEMATICS.
Analytic Geometry, Combinations & Permutations, Complex Numbers, Congruences, Fractions, Large Numbers (to 230 digits), Linear Equations & Matrices (to N=36), Polynomials Y=F(X) (to degree 50), Prime Factors, Trigonometry, and MORE! *VERY user friendly.* These give rational results (instead of decimal) wherever possible. Currently for 128K 80-column Apple II Series.

**MATH WORKHORSE $79.95**

Specify 5.25 or 3.5 inch format (2 disks/1 disk), or send $1.00 for a complete description of these programs.

**Spence Barnshaw, Box 35032, Vancouver, B.C. V6M 4G1 (CANADA)**

### EVERGLADE

**a poem in hypertext**
**by Rod Willmot**

Programmed by its author, a Canadian poet with an international audience, **Everglade** opens the door to new forms of literature. A first in IBM-compatible shareware, PC-SIG disk #1901. To order fully registered send $9.95 and disk size to: **Hyperion SoftWord**
535 Duvernay
Sherbrooke, QC,
CANADA J1L 1Y8

**RECREATIONAL & EDUCATIONAL COMPUTING**

**By Recreational Computing Columnist/ Math Professor Dr. Michael Ecker,**
*with contributed columns by Martin Gardner and others*

Send $24.00 for one year of 8 Big Issues, or SASE for more information to:
**The REC Newsletter**
**129 Carol Drive**
**Clarks Summit, PA 18411**

## LETTERFORMS & ILLUSION

**Scott Kim and Robin Fe Sameleson**

This set of computer programs is a collection of entertaining images, activities, puzzles and fonts that invites readers to explore for themselves the fascinating and challenging world of visual perception.

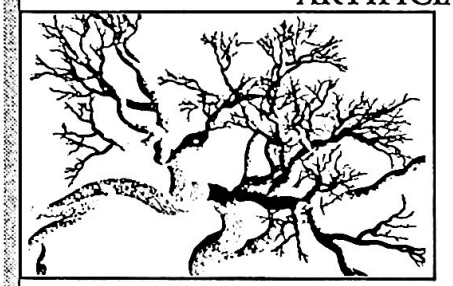
Designed for use with MacPaint™, the box consists of completely self-contained 3 1/2" disk and a 48-page book which guides the reader through the programs and projects. The simple programs of **LETTERFORMS & ILLUSIONS** will prove personally rewarding to the computer enthusiast or general reader with an interest in geometry, art and design. **ISBN 0-7167-2043-4, $39.95**

**W.H. Freeman and Company**
41 Madison Avenue New York, NY 10010

**BELM** *creates*
checkerboard worlds and
writes about them as they
evolve. Unique, non-random
binary-English translation.
5 1/4" diskette
(IBM-PC) $25.00
Sample story $2.00

B.V. Firner
415 Lancaster Ct.
Piscataway, NJ 08854

ARTIFICIAL LIFE (The Book)
edited by
Christopher G. Langton



May be ordered by calling
1-800-447-2226
ask for
"09356" paperback
or "09346" hardcover

Addison-Wesley Publishing Company, Inc.

**NEED CORE WAR SOFTWARE?**
**WANT THE BEST!?**

If you have an IBM PC or compatible, at least 256K of RAM, and a CGA Display (or emulator), AMRAN has the Core War Software of your dreams!

For prompt delivery, send $34.95 U.S. to:
AMRAN, 5712 Kern Drive
Huntington Beach, CA 92649-4535
Core War Coliseum, Recode Assembler, sample Warrior Programs, and 40 page manual included on single 360K floppy disk.

MegageM
Presents
FractalPro

The Premier Fractal Art & Animation System For All Amiga PCs
Mandelbrot & Julia Sets And Their 'Cube' Analogs in 256 Colors
Magnify up to 10 Trillion - Color Palette Control With Color Cycling
Automatic Sequencing for Superb Smooth Animated Zooms and Pans
Six Directions of Motion : Zoom (In-Out) and Pan (Left-Right-Up-Down)
Handcrafted FAST Assembly Language - Transparent 68020/30 Support
Interactive Full Featured User Interface - Free Tech Support
Only $89.95 + $2.00 Shipping (6.25% Tax in California) - Sample Pics & Catalog on Disk : $2.50
MegageM 1903 Adria Santa Maria, CA 93454 (805) 349-1104

MATH
ALGORITHM
√Σ%∫∂¥Ø∏ LIBRARY

AT LAST: A compact easy, yet very powerful take-home Maths library. For students or experts who simply MUST solve nasty problems rapidly but who don't have time. Friendly, quickstart MAL is a lifelong investment: 45 MALgorithms - polynomials, plots, fourier transforms, eigen values, primes and much more...

WRITE: MALco
Con Street, Macleay Is, Queensland, AUSTRALIA 4184


## BULLETIN BOARD

*DEADLINES:*

All Bulletin Board items for the next issue, Algorithm 1.5, should be received by us no later than **May 31, 1990.**

Mail to: **Algorithm**
**2 Oxford Street, West**
**London, Ontario N6H 1P9**

or fax to: (519) 432-8778


PUZZLEPACK - $20.00 US
Solve jumbled words, anagrams, crosswords.
Find all words, any length, in a set of letters.
Fill in "wild card" blanks, fixed or floating.
96,000-word RAM dictionary, amazing speed!
Requires 640K PC compatible or PCjr.

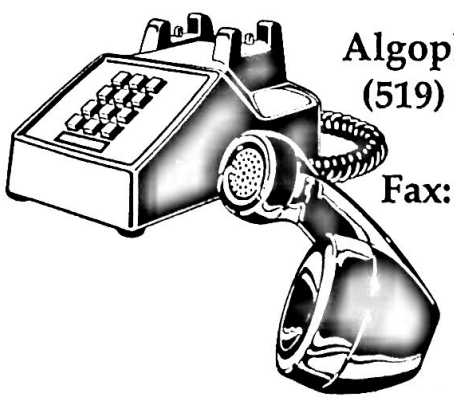
PERMTABL - $15.00
Permutation tables in unique cyclic form.
Up to 16 elements, including duplicates. Sets which generate more than 3 pages of cycles are skipped.
130-page printable file plus generating source (BASICA).

Specify 5 1/4" or 3 1/2" floppies. Check or M.O.

RECROOM RECWARE
P.O. Box 307, Pacific Grove, CA 93950


## LATE MAIL!

Algorithm 1.3 entered the U.S. Postal system on or about March 2, 1990. If you have subscribed and did not receive this issue (or any other issue), please inform us and we will promptly send you a replacement.



**Algophone:**
**(519) 432-8042**

**Fax: (519) 432-8778**

# COPY-THIS PAGE

# WRITE US A LETTER

*Moving? Have a product or service to distribute? Wish a friend to subscribe? Want to suggest a topic?* Fill in the appropriate space below and mail the copy to us at:

Algorithm
P.O. Box 29237 Westmount Postal Outlet
785 Wonderland Road South
London, Ontario, Canada N6K 1M6

Name .....

| Old address | New address |
|---|---|
| | |
| | |
| | |
| | |

effective date: .....

If you're moving, let us know by filling out the space on the left. This way you can ensure an uninterrupted flow of fascinating projects and engaging articles.

Perhaps you know someone who would like to subscribe to Algorithm, the Personal Programming Newsletter. If so, copy this page and fill in the space on the right.

*PLEASE* send me 6 issues of Algorithm for the introductory price of **$29.95**

NAME:_____

STREET:_____

CITY:_____

STATE:_____

COUNTRY:_____ ZIP:_____

PLEASE SEND PAYMENT TO:
Algorithm P.O. Box 29237, Westmount Postal Outlet, 785 Wonderland Rd., London, Ontario, Canada N6K 1M6
(Cheque or money order. No C.O.D.)
Can. Funds $35.95 • Overseas add $4.00 Shipping.

If you have a product or service you'd like to make available to our readers, fill out the space to the left with a rough sketch of your ad and we will set it in type. First appearance: FREE! Subsequent appearances: $20.00/issue.

*Is there a topic you'd like to see us cover?* Fill out the details in the space below:

Dear Editor: _____

_____

_____

_____

_____

_____

Name: _____ Phone: _____

Address: _____ Date: _____